

WORK SUPPORTED BY
NASA-NASA 191969
191969-191969

UNIVERSITY OF CALIFORNIA

Los Angeles

IN 61/11

191969
F-174

Reliable and Efficient Parallel Processing

Algorithms and Architectures

for Modern Signal Processing

N93-16723

Unclass

G3/61 0135966

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy in Electrical Engineering

By

KuoJuey Ray Liu

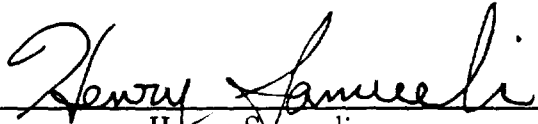
1990

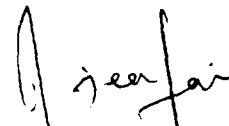
(NASA-CR-191969) RELIABLE AND
EFFICIENT PARALLEL PROCESSING
ALGORITHMS AND ARCHITECTURES FOR
MODERN SIGNAL PROCESSING Ph.D.
Thesis (California Univ.) 174 p


CAST

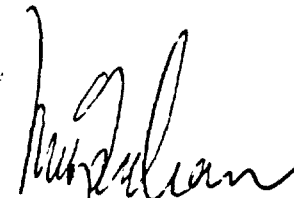
© Copyright by
KuoJuey Ray Liu
1990

The dissertation of KuoJuey Ray Liu is approved.


Henry Samueli


Rajeev Jain


Milos Ercegovac


Tony Chan


Kung Yao
Committee Chair

University of California, Los Angeles

1990

Dedication

To my wife Ching-Ling,
who shared this experience with love, understanding, and support.

To my great-grand mother Lang-Shuang
for her kindly love since my childhood.

To my son Jeffry
for the joyful time that we share together.

Contents

Dedication	iii
List of Figures	ix
List of Tables	x
ACKNOWLEDGEMENTS	xi
VITA	xii
ABSTRACT	xiv
1 Introduction	1
2 QRD RLS Algorithms Using Householder Transformation	8
2.1 QRD Recursive Least-Squares Algorithm	9
2.2 Systolic Block Householder Transformation	15
2.2.1 Vectorized SBHT QRD Systolic Array	19
2.3 SBHT RLS Algorithm	20
2.3.1 Vectorized SBHT RLS Array	26
2.4 Two-level Pipelined Implementations	30

2.5	Constrained RLS Problems	36
3	Real-Time Algorithm-Based Fault-Tolerance for QRD RLS Systolic Array	41
3.1	Algorithm-Based Fault-Tolerance	42
3.2	Fault Model	46
3.3	Real-Time Fault-Tolerance	47
3.3.1	Concurrent Error Detection - Residual Method	47
3.3.2	Fault Diagnosis	57
3.3.3	Order-Degraded Reconfiguration	62
3.4	Error Propagation and Recovery Latency	63
3.4.1	Robust Error Detection	64
3.4.2	Latencies	67
3.5	Conservation Test	69
4	Dynamic Range, Stability, and Fault-tolerant Capability of Finite-precision QRD RLS Systolic Algorithm	72
4.1	Quasi Steady-State and Ensemble Behavior	73
4.2	Dynamic Range and Lower Bound on Memory Size	78
4.3	Stability and Quantization Effect	86
4.4	Finite-length Effect of Fault-tolerant Capability	89
4.4.1	Missing Error Detection	89
4.4.2	False Alarm	92
4.4.3	Overall Memory Size Consideration	96
5	Order Degraded Performance and Residual Estimations	98

5.1	Order Degraded Performance	98
5.1.1	Geometric Interpretation	103
5.2	Residual Estimation in Faulty Situation	104
5.2.1	Faulty Internal Cell	105
5.2.2	Faulty Boundary Cell	108
6	Multi-phase Systolic Algorithms for Spectral Decomposition	109
6.1	Recent Developments	110
6.2	Systolic Array Matrix Processing	114
6.3	QR Algorithm	120
6.4	Multi-phase Systolic Algorithms	124
6.4.1	Multi-phase Triangular Systolic Array	125
6.4.2	Multi-phase Rectangular Systolic Array	130
6.4.3	The Hessenberg Reduction	133
6.4.4	Computing the Eigenvectors	136
6.5	Performance Efficiency	138
6.5.1	Comparisons of the arrays	138
6.5.2	Rate of Convergence	139
6.6	Efficient Fault-tolerance Schemes	141
7	Conclusions and Future Research	146

List of Figures

2.1	QRD RLS systolic array	13
2.2	Processing Cells of QRD RLS systolic array	14
2.3	The SBHT QRD systolic array.	21
2.4	The processing cells of the SBHT QRD systolic array.	22
2.5	The SBHT RLS systolic array obtained by direct generalization of the Givens rotation array.	27
2.6	The SBHT RLS systolic array.	29
2.7	Operations of the processing cells by using modified Householder trans- formation.	32
2.8	Processing cells for two-level pipelined implementation.	33
2.9	Operations of the processing cells	34
2.10	The MVDR beamforming systolic array	40
3.11	Matrix-matrix multiplication and LU decomposition with (weighted) checksum encoding	44
3.12	Recursive LS systolic array with multiple desired responses	48
3.13	Fault-tolerant recursive LS systolic array based on linear combination of input data in top row array feeding into column error detection array with output fault indication variable e_0	52

3.14	Processing cells of fault-tolerant systolic array.	53
3.15	Plot of $ e_0 $ of an adaptive QRD LS filter with order $p = 3$ when a fault occurred in PE_{23} from $t = 25$ to $t = 35$	55
3.16	Plot of $ e_0 $ in Fig.3(a) with threshold set at 0.3.	55
3.17	Plot of the hardware efficiency of the residual method versus the order of the LS estimation.	57
3.18	Reduced $(p - 1) \times (p - 1)$ tri-array after the deletion of the row and column with a faulty cell.	63
3.19	Plot of expected processing latency and expected error propagation latency versus the order of the LS estimation.	68
3.20	Comparisons of expected recovery latency for FFL and CSE methods.	70
4.21	Plots of the variances in dB	78
4.22	Contents of the Cells	82
4.23	The first two rows of the array	93
5.24	Geometric illustration	104
6.25	Triangular systolic array for QR decomposition.	115
6.26	Computation of $R^{-T}\mathbf{x}$ using a triarray.	119
6.27	Multiplication of a triangular matrix and a full dense matrix.	120
6.28	Matrix-matrix multiplication in a rectangular array.	121
6.29	A circular multiplexer.	124
6.30	A first in/first out buffer.	125
6.31	Phase 1: The QR decomposition.	127
6.32	Phase 2: Computing the Q matrix.	128
6.33	Phase 3: Computing the matrix product RQ	129

6.34	Multi-phase rectangular array for the QR iteration.	131
6.35	System configuration of the multi-phase triarray.	137
6.36	System configuration of the multi-phase rectangular array.	137
6.37	The number of iterations for a QR algorithm to converge versus the matrix size.	140
6.38	Row checksum for $A_r = QR_r$	143
6.39	Column checksum for $R^{-T} A_c^T = Q_c^T$	144

List of Tables

2.1	Comparisons of the SBHT and Givens rotation methods	36
3.2	The output of e_0 and e of an adaptive QRD LS filter	54
4.3	Mean distribution for different input data with different λ values . . .	76
4.4	Variance distributions for different input data with different λ values	77
6.5	Operations of the processing cells for different phases.	116
6.6	The timing table for the rotation parameters to reach the right edge of the QR triarray.	117
6.7	Comparisons of the multi-phase triarray and rectangular array. . . .	138

Acknowledgments

I would like to thank Professor Kung Yao for his counsel, comments, and constant encouragement on my research, especially his time and effort on reading and revising many papers that we published. I also would like to thank the members of my doctoral committee for their time spent in reviewing the dissertation research.

I am grateful to Professor Ken Martin for his kindness to provide me the working environment in ICSL in which many of my research were done. I would like to thank S.F. Hsieh for various helpful discussions that led to many research ideas. Also, I

This work was partially supported by the NASA/Ames under Grant 2-374, the National Science Foundation under Grant NCR-8814407, and the UC MICRO Grant.

Finally, I cannot adequately express my deeply appreciation to my wife, Ching-Ling, for her constant support and love during these years of graduate study, especially those days that we can only buy 30c toast with an almost empty bank account. I also appreciate my parents for their encouragement and love.

VITA

KuoJuey Ray Liu

	Born in
1983	B.S. , Electrical Engineering Department, National Taiwan University, Taipei, Taiwan
1983-1985	Communication Officer, Signal Corp. Taiwan
1986-1987	Teaching Assistant, University of Michigan
1987	Fifth Annual VLSI Design Contest - Fifth prize: VLSI implementation of a median filter
1987	M.S.E. , Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor
1987	President Research Partnership, Office of the President, University of Michigan
1987-1988	University Fellowship, UCLA
1988-1990	Teaching Associate, UCLA
1989	Hortense Fishbaugh Memorial Scholarship, UCLA
1989	Graduate Student Award in Science and Engineering, Taiwanese-American Foundation

PUBLICATIONS AND PRESENTATIONS

1. K.J.R. Liu and K. Yao "On uniform one-chip VLSI design considerations for some discrete orthogonal transforms", Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) pp.2136-2139, New York, April 1988.
2. K.J.R. Liu and K. Yao, "A systematic approach to bit recursive systolic array design" Proc. IEEE International Conference on Systolic Array, pp.685-694, San Diego, May 1988.
3. K.J.R. Liu, "Optimal Graph-Theoretical Layout of CMOS Functional Cells" Tech. Report, UCLA-ENG 88-29, Sep. 1988.

4. K.J.R. Liu and K. Yao, "Gracefully degradable real-time algorithm-based fault-tolerant method for QR recursive least-squares systolic array", in *Systolic Array Processors*, Ed. McCanny, McWhirter, and Swartzlander, pp.401-410, Prentice Hall (UK), 1989.
5. S.F. Hsieh, K.J. R. Liu, and K. Yao, "A fast and effective algorithm for sinusoidal frequency estimation", *IEEE Int'l Symposium on Information Theory*, San Diego, Jan. 1990.
6. K.J.R. Liu and K. Yao, "Spectral decomposition via systolic triarray based on QR iteration", *Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing (ICASSP)*, pp.1017-1020, Albuquerque, April 1990.
7. S.F. Hsieh, K.J. R. Liu, and K. Yao, "Applications of truncated QR methods to sinusoidal frequency estimation", *Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing (ICASSP)*, pp.2571-2574, Albuquerque, April 1990.
8. K.J.R. Liu, S.F. Hsieh, and K. Yao, "Recursive LS filtering using block Householder transformation" *Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing (ICASSP)*, pp. 1631-1634, Albuquerque, April 1990.
9. K.J.R. Liu, "Dynamic range for finite-precision QRD LS algorithm and its stability", *Proc. Int'l Sym. Circuits and Systems (ISCAS)*, pp.3142-3145, New Orleans, May 1990.
10. S.F. Hsieh, K.J.R. Liu, and K. Yao, "Comparisons of truncated QR and SVD methods for sinusoidal frequency estimation", *Proc. The second Int'l Workshop on SVD and Signal Processing*, Kingston, June 1990.
11. K.J.R. Liu and K. Yao, "Multi-phase systolic architectures for spectral decomposition" *Proc. Int'l Conf. on Parallel Processing*, August, 1990.
12. K.J.R. Liu, S.F. Hsieh, and K. Yao, "Two-level pipelined implementation of systolic block Householder transformations with application to RLS algorithm" *Proc. Int'l Conf. on Application-Specific Array Processors*, Princeton, Sep. 1990.
13. K.J.R. Liu, S.F. Hsieh, and K. Yao, "Performance comparisons of parallel SVD in VLSI array processors", *VLSI Signal Processing IV*, San Diego, Nov. 1990.

ABSTRACT OF THE DISSERTATION

**Reliable and Efficient Parallel Processing
Algorithms and Architectures
for Modern Signal Processing**

by

KuoJuey Ray Liu

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 1990

Professor Kung Yao, Chair

Least-squares (LS) estimations and spectral decomposition algorithms constitute the heart of modern signal processing and communication problems. Implementations of recursive LS and spectral decomposition algorithms onto parallel processing architectures such as systolic array with efficient fault-tolerant schemes are the major concerns of this dissertation.

There are four major results in this dissertation. First, we propose the systolic block Householder transformation with application to the recursive least-squares minimization. It is successfully implemented on a systolic array with a two-level pipelined implementation at the vector level as well as at the word level.

Second, a real-time algorithm-based concurrent error detection scheme based on the *residual method* is proposed for the QRD RLS systolic array. The fault diagnosis, order degraded reconfiguration, and performance analysis are also considered.

Third, the dynamic range, stability, error detection capability under finite-precision implementation, order degraded performance, and residual estimation under faulty situations for the QRD RLS systolic array are studied in details.

Finally, we propose the use of multi-phase systolic algorithms for spectral decomposition based on the QR algorithm. Two systolic architectures, one based on triangular array and another based on rectangular array, are presented for the multi-phase operations with fault-tolerant considerations. Eigenvectors and singular vectors can be easily obtained by using the multi-phase operations. Performance issues are also considered.

Chapter 1

Introduction

In order to meet the demand of high throughput and high computational complexity of modern signal processing, parallel processing algorithms and architectures have been extensively studied and implemented for dedicated applications. Rapid advances in VLSI microelectronics make it practical to build low-cost and high-density application-specific integrated circuits (ASIC) to meet the demands of speed and performance of modern signal processing. Recent VLSI/WSI technology permits the building of million of transistors in a single chip, while a large system may require hundreds of these chips to function properly. For a complex system, a single fault from any part of the system can make the whole system useless. For various critical applications, highly-reliable computations are demanded. Fault-tolerance is therefore needed in many of these problems. From these reasons, there is a potential proliferation of activities in the new area of fault-tolerant signal processing which explores reliable ways to implement fast and efficient signal processing algorithms. The goal of this dissertation is to study the parallel processing algorithms and architectures as well as associated efficient fault-tolerant techniques for modern signal processing.

Least-squares (LS) problems and spectral decomposition algorithms constitute the heart of modern signal processing and communications applications such as adaptive filtering, spectral estimation, array signal processing, channel equalization, etc.. Implementations of recursive LS and spectral decomposition algorithms onto parallel processing architectures such as systolic array with efficient fault-tolerant schemes are the major concerns of this dissertation.

Efficient implementation of the LS algorithm, particularly the recursive LS algorithm (RLS), is needed to meet the high throughput and speed requirements of modern signal processing. There are many possible approaches such as fast transversal method and lattice method which can perform RLS algorithm efficiently [6, 29]. Unfortunately, these methods can encounter numerical difficulties due to the accumulation of round-off errors under a finite-precision implementation as summarized in [18]. This may lead to a divergence of the computations of the RLS algorithm [18]. A new type of systolic algorithm based on the QR decomposition (QRD) known as the QRD RLS was first proposed by McWhirter in [72]. This algorithm is one of the most promising algorithms in that it is numerical stable [6, 54] as well as suitable for parallel processing implementation on a systolic array [29, 72].

Up to now, most of the QRD RLS implementations were based on the Givens rotation method and modified Gram-Schmidt method which both are rank-1 update approaches [19, 26, 31, 56, 67, 72, 43]. It is well-known that the Householder transformation (HT), which is a rank- k update approach, is one of the most computationally efficient methods to compute QRD. The error analysis carried out by Wilkinson [97, 39] showed that the HT outperforms the Givens method under finite precision computations. Presently, there is no known technique to implement the HT on a

systolic array parallel processing architecture, since there is a belief that non-local communications in the implementation are necessary due to the vector processing nature of the Householder transformation. One of the purposes of this paper is to show that we can implement the HT on a systolic array with only local connections. Thus, it is amenable to VLSI implementation and is applicable to real-time high throughput applications of modern signal processing.

In Chapter 2, we first propose a systolic Householder algorithm called a systolic block Householder transformation (SBHT) to compute the QRD with an implementation on a vectorized systolic array. Then a RLS algorithm based on the SBHT called SBHT RLS algorithm is proposed to perform RLS operations on the array. We shall show that the SBHT array and the SBHT RLS array are generalizations of Gentleman-Kung's QRD array [26] and McWhirter's QRD RLS systolic array [72] respectively. The difficulty in the applications of the above arrays is mainly due to the the vectorized operations of the processing cells. This results in a high cell complexity as well as a high I/O bandwidth. By using a modified HT algorithm proposed by Tsao [95], a two-level pipelined implementation of the SBHT RLS algorithm can be achieved. That is, the algorithm is pipelined at the vector level as well as at the word level. The complexity of the processing cell as well as the I/O bandwidth are thus reduced. In general, the cell complexity of the SBHT array is higher and the system latency is longer than that of the conventional Givens rotation implementations. With the two-level pipelined implementation, the throughput of the SBHT RLS systolic array is as fast as that of McWhirter's Givens rotation array, and it offers better numerical property than the Givens method. In addition, an extension of the SBHT RLS array to MVDR beamformation, which is a constrained RLS problem, is also

considered.

Fault-tolerance has been defined as *the ability of a system to execute specified algorithms correctly regardless of hardware failures and program errors* [?]. In order to achieve the goal of fault-tolerance, redundancy has to be introduced. When we encounter a specific VLSI signal processing problem, an inherent nature of that signal processing algorithm can be used to develop a highly efficient specific fault-tolerant technique named *algorithm-based fault-tolerance*. In Chapter 3, we propose a new algorithm-based fault-tolerant scheme derived from the inherent nature of the QRD LS systolic algorithm called the *residual method*. For a LS problem, especially in communication and signal processing applications, we abstract information from the residuals which are the differences of the optimal LS estimations from input data and the desired responses. Based on the fact that a QRD LS systolic array can compute the residuals of different desired responses simultaneously, an artificial desired response can be designed to detect any error produced by a faulty processor. We show that if the artificial desired response is designed as a non-zero linear combination of all data inputs, the residual output of this response will be zero if no fault occurred. Any fault in the system will cause the residual to be non-zero and thus the fault is detected in real-time. Thus, the residual method can be easily incorporated with the systolic array antenna beamforming systems such as that considered in [77] and will have a great impact on the next generation of radar and sonar systems where the fault-tolerance scheme is quite essential for reliable real-time operation.

Once the fault has been detected, two methods to diagnose the location of the faulty row are addressed. The first method, called the flushing fault location method, is based on the weight flushing technique to flush the weight vector out by using an

identity matrix input. The second method is to use the checksum encoding property to detect the row which does not meet the checksum condition. When the faulty row is determined, this row and the column associated with the same boundary cell are eliminated by a reconfiguration operation. Then the system operates in an order-degraded manner which is acceptable in many least-squares applications.

Though the QRD RLS algorithm is generally recognized as having good numerical properties such as numerical stability under finite-precision implementation [8, 60], there is no detailed study of this until a recent paper by Leung and Haykin [60]. Presently, it is still not known how to obtain the dynamic range of the algorithm in determining the word-length to ensure correct operation of the algorithm.

In Chapter 4, we first observe that the cosine parameters generated by boundary cells will eventually reach the *quasi steady-state* if λ is close to one which is generally the case. We will show that the quasi steady-state and ensemble values of sine and cosine parameters are the same for all boundary cells. It is independent of the statistics of the input data sequence and the position of the boundary cell which generates the sine and cosine parameters. Simulation results are presented to support this observation. These results yield the tools needed to further investigate many properties of the QRD LS systolic algorithm. Then, we can obtain upper bounds on the dynamic range of the processing cells. Thus, lower bounds on the memory size can be obtained from these upper bounds on the dynamic range to prevent overflow and to ensure correct operations of the QRD LS algorithm. With these results, we reconsider the stability problem under quantization effects with a more general analysis and obtain tighter bounds than given in previous work [60]. Two important factors of the fault-tolerant capability, the *missing error detection* and the *false alarm* effects

are also studied in this chapter.

When the faulty row is found in the fault-tolerant QRD LS systolic array, we enter an order degraded operation. In Chapter 5, we study the performance degradation when the order of the LS is reduced and give a geometric interpretation. For a short-time transient fault, we do not switch to the order degraded operation. Thus we propose an approximate method to estimate the optimal residual in this faulty situation.

Computing the spectral decomposition of a matrix is an important issue in many modern signal processing and system applications. The feasibility of real-time processing for sophisticated modern signal processing systems, depends crucially on efficient implementation of parallel processing of the algorithms and associated architectures needed to perform these operations [14, 58]. While many variations exist in the literatures for solving these matrix problems, the heart of all these iterative methods are based either on the Jacobi-Hestennes method or the QR algorithm [30, 101, 107]. While there are some fundamental differences between these two approaches, both algorithms have good numerical stability and convergence rate properties and thus are desirable for possible implementation. Since present VLSI technology is capable of building a multiprocessor system on a chip, many researchers have proposed different parallel processing architectures to solve eigenvalue and singular value decomposition (SVD) problems.

Presently, there is no known simple efficient systolic array approach for the generation of eigenvectors. The main reason is that there is no single architecture that is capable of handling all the steps required in the algorithm such that we can pipeline

the successive iterations readily. The communication cost among different architectures is high and the interface problem for an efficient data flow is demanding. In this paper, we propose two multi-phase systolic algorithms to solve the spectral decomposition problem based on the QR algorithm. By multi-phase operations we mean that the processing cells can perform different arithmetical operations in different phase of the computations. Two systolic arrays, one is triangular and the other is rectangular, are designed based on the multi-phase concept. A key feature in our method for the successfully application of the QR algorithm is that the Q matrix of the QR decomposition can be computed explicitly by multiphase operations. With the proper feedback of this Q matrix, the QR algorithm can be computed and pipelined effectively in a single systolic array. From the accumulation of those Q matrices in another array, eigenvectors and singular vectors can be computed without needing global communication inside the array. All these issues are considered in Chapter 6.

Finally, Chapter 7 summarizes the research results of this dissertation and provides future research directions in these areas.

Chapter 2

QRD RLS Algorithms Using Householder Transformation

The QRD RLS algorithm is one of the most promising RLS algorithms, due to its robust numerical stability and suitability for VLSI implementation based on a systolic array architecture. Up to now, among many techniques to implement the QR decomposition, only the Givens rotation and modified Gram-Schmidt methods have been successfully applied to the development of the QRD RLS systolic array. It is well-known that Householder transformation (HT) outperforms the Givens rotation method under finite precision computations. Presently, there is no known technique to implement the HT on a systolic array architecture. In this chapter, we propose a *Systolic Block Householder Transformation* (SBHT) approach, to implement the HT on a systolic array as well as its application to the RLS algorithm. Since the data is fetched in a block manner, vector operations are in general required for the vectorized array. However, by using a modified HT algorithm, a two-level pipelined implementation can be used to pipeline the SBHT systolic array both at the vector

and word levels. The throughput rate can be as fast as that of the Givens rotation method. Our approach makes the HT amenable for VLSI implementation as well as applicable to real-time high throughput applications of modern signal processing. The constrained RLS problem using the SBHT RLS systolic array is also considered in this chapter.

In section 2.1, a brief review of the QRD RLS algorithm is given. In section 2.2, the SBHT is presented while the SBHT RLS algorithm is considered in section 2.3. The two-level pipelined implementation of the SBHT RLS systolic array is discussed in section 2.4. Finally, in section 2.5, the constrained RLS problem, as applied to MVDR beamformation, using an extension of the SBHT RLS array is presented.

2.1 QRD Recursive Least-Squares Algorithm

Consider a $n \times p$ real-valued data matrix $A(n)$ denoted by

$$A(n) = [\underline{u}(1), \underline{u}(2), \dots, \underline{u}(n)]^T = [\underline{a}(1), \underline{a}(2), \dots, \underline{a}(p)], \quad (2.1)$$

a $n \times 1$ desired response vector $\underline{y}(n) = [d(1), d(2), \dots, d(n)]^T$, a $p \times 1$ weight vector $\underline{w}(n)$, and a $n \times 1$ residual vector

$$\underline{\epsilon}(n) = [e(1), e(2), \dots, e(n)]^T = A(n)\underline{w}(n) - \underline{y}(n).$$

Let the index of performance be defined by the weighted l_2 norm of

$$\xi(n) = \|\underline{\epsilon}(n)\|_{\Lambda}^2 = \|\Lambda \underline{\epsilon}(n)\|^2 = \underline{\epsilon}^H(n) \Lambda^2(n) \underline{\epsilon}(n),$$

where $\Lambda(n) = \text{diag}[\lambda^{(n-1)}, \lambda^{(n-2)}, \dots, \lambda, 1]$ with a real-valued forgetting factor $0 < \lambda \leq 1$. Then the LS solution, satisfies

$$\xi_{\min}(n) = \min_{\underline{w}} \|A(n)\underline{w}(n) - \underline{y}(n)\|_{\Lambda}^2 = \|A(n)\hat{\underline{w}}(n) - \underline{y}(n)\|_{\Lambda}^2. \quad (2.2)$$

Assume A to be full rank with $n \geq p$. Then the QR Decomposition of $\Lambda(n)A(n)$ yields

$$Q(n)\Lambda(n)A(n) = [R(n)^T, \mathbf{0}]^T,$$

where $R(n)$ is a $p \times p$ upper triangular matrix and $Q(n)$ is an $n \times n$ unitary matrix.

Thus

$$\xi(n) = \|Q(n)\underline{e}(n)\|_\lambda^2 = \|[R(n)\underline{w}(n) - \underline{P}(n)]\|^2 + \|\underline{v}(n)\|^2,$$

where

$$[\underline{P}^T(n), \underline{v}^T(n)]^T = Q(n)\Lambda(n)\underline{y}(n),$$

with a $p \times 1$ vector $\underline{P}(n)$ and a $n \times 1$ vector $\underline{v}(n)$. The LS solution $\hat{\underline{w}}(n)$ can be obtained from

$$R(n)\hat{\underline{w}}(n) = \underline{P}(n). \quad (2.3)$$

Then $\xi_{\min}(n) = \|\underline{v}(n)\|^2$.

For some radar/sonar and communication problems, the weight vector $\underline{w}(n)$ may not be of direct interest. For example, the residual is of interest in the multiple sidelobe canceller adaptive array problem. Let $\underline{u}(n)$ be the n^{th} new incoming row vector of data in $A(n)$ of (2.1). Then the LS residual

$$e(n) = \underline{u}^T(n)\hat{\underline{w}}(n) - d(n), \quad n = p, p+1, \dots, \quad (2.4)$$

can be obtained without computing $\hat{\underline{w}}(n)$ explicitly. Thus, the complexity of the LS problem is further reduced and the solution of $e(n)$ is feasible with a systolic array implementation [78].

To compute the linear LS problem recursively, a unitary matrix $\bar{Q}(n-1)$ is defined as

$$\bar{Q}(n-1) = \left[\begin{array}{c|c} Q(n-1) & \underline{0}_{n-1} \\ \hline \underline{0}_{n-1}^T & 1 \end{array} \right]$$

and we have

$$\bar{Q}(n-1)\Lambda(n)A(n) = \left[\begin{array}{c} \lambda R(n-1) \\ \hline \mathbf{0} \\ \hline \underline{u}^T(n) \end{array} \right].$$

Suppose the Givens rotation method is used for the QRD, then $\underline{u}^T(n)$ can be annihilated by applying a sequence of Givens rotations

$$G(n) = G_p(n) \cdots G_2(n)G_1(n), \quad (2.5)$$

where the $n \times n$ transformation matrix G_i is defined by

$$G_i = \left[\begin{array}{cccccc} I_{i-1} & \vdots & 0 & \vdots & 0 & \vdots & 0 \\ \hline & & c_i & & 0 & & s_i \\ \hline & & & & & & \\ 0 & \vdots & 0 & \vdots & I_{n-i-1} & \vdots & 0 \\ \hline & & & & & & \\ 0 & \vdots & -s_i & \vdots & 0 & \vdots & c_i \end{array} \right],$$

with

$$c_i = \frac{a}{\sqrt{a^2 + b^2}}, \quad s_i = \frac{b}{\sqrt{a^2 + b^2}},$$

where a and b are elements of vectors in the i^{th} and n^{th} rows under rotation. The matrix $G(n)$ can be shown to have the form

$$G(n) = \left[\begin{array}{c|c|c} K(n) & \mathbf{0} & \underline{h}(n) \\ \hline \mathbf{0} & I_{n-p-1} & \mathbf{0} \\ \hline \underline{h}^H(n) & \mathbf{0} & \gamma(n) \end{array} \right],$$

where $K(n)$ is a $p \times p$ matrix, $\underline{h}(n)$ is a $p \times 1$ vector, I_{n-p-1} is the $(n-p-1) \times (n-p-1)$ identity matrix, and $\gamma(n)$ is a scalar given by $\gamma(n) = \prod_{i=1}^p c_i(n)$, $n \geq p$ with $c_i(n)$ is the cosine parameter associated with the i^{th} Given rotation. The error vector can be transformed by $\bar{Q}(n-1)$ to

$$\bar{Q}(n-1)\Lambda\underline{\epsilon}(n) = \begin{bmatrix} \lambda R(n-1) \\ \hline \mathbf{0} \\ \hline \underline{u}^H(n) \end{bmatrix} \underline{w}(n) - \begin{bmatrix} \lambda \underline{P}(n-1) \\ \hline \lambda \underline{v}(n-1) \\ \hline d(n) \end{bmatrix}.$$

When the Given rotation is applied on it, we get

$$G(n)\bar{Q}(n-1)\Lambda\underline{\epsilon}(n) = \begin{bmatrix} R(n) \\ \hline \mathbf{0} \end{bmatrix} \underline{w}(n) - \begin{bmatrix} \underline{P}(n) \\ \hline \underline{v}(n) \end{bmatrix}.$$

It can be easily seen that the last element of $\underline{v}(n)$, $v_n(n)$, can be obtained naturally during the triangularization process [78] and is given by

$$v_n(n) = \lambda \underline{h}^H(n) \underline{P}(n-1) + d(n)\gamma(n).$$

In [78], McWhirter has shown the LS residual $e(n)$ can be expressed as

$$e(n) = v_n(n)\gamma(n).$$

The above results lead to the systolic implementation of QRD RLS algorithm without computing weight vector explicitly [78]. The systolic array is shown in Fig. 2.1. It consists of two parts: a triangular array for computing QRD and a linear column array called response array (RA) for computing LS residual. When a new data vector is updated, a new triangular matrix R sits in the triangular QR array and a new vector \underline{P} sits in the RA.

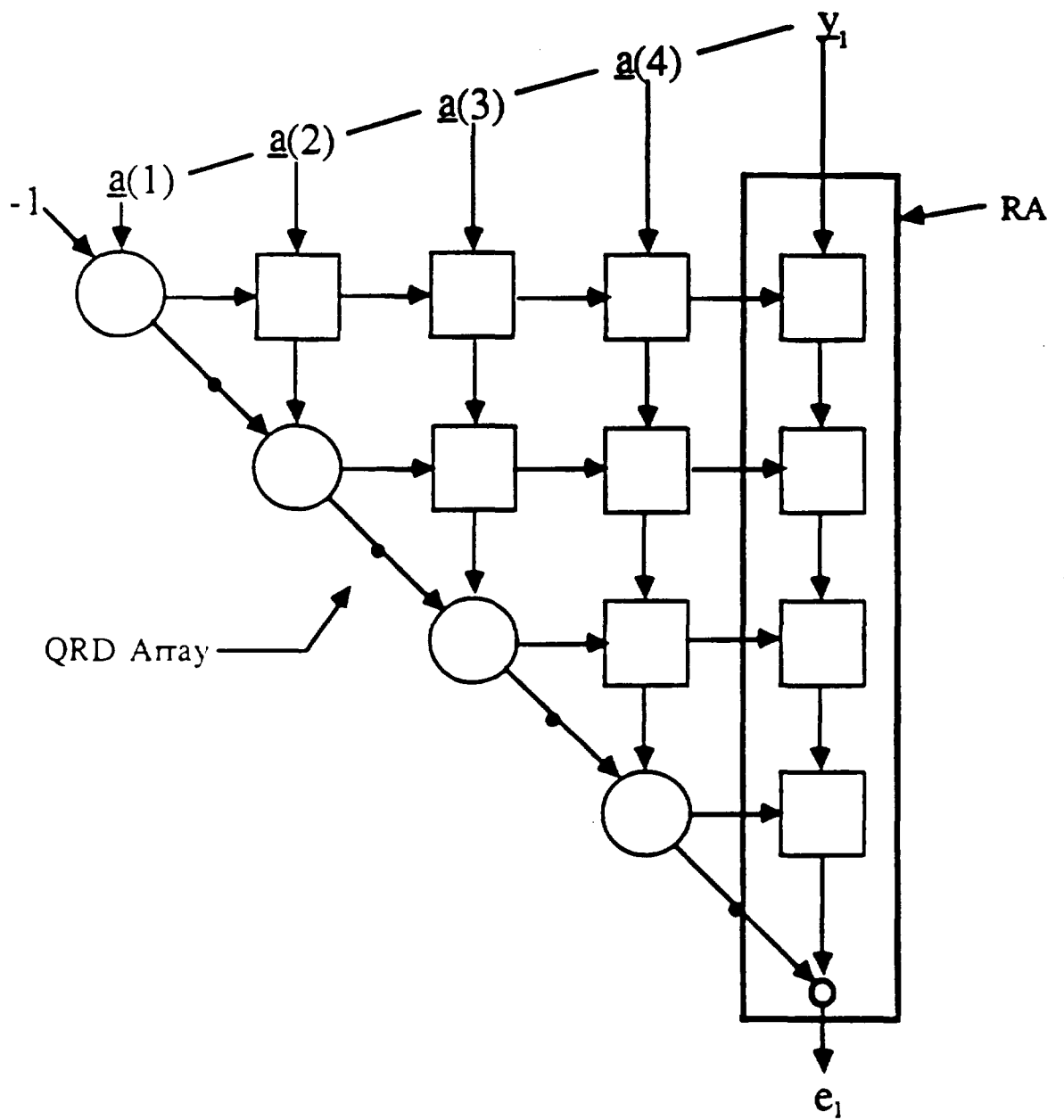
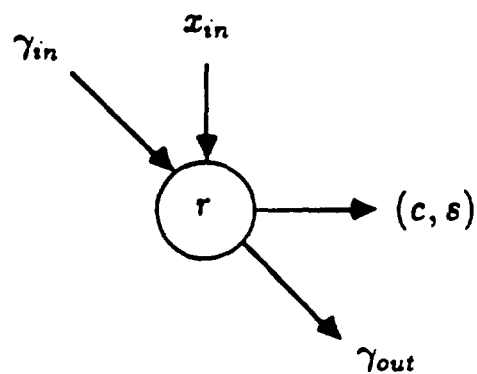


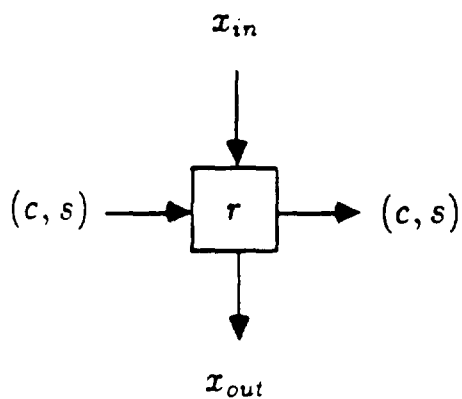
Figure 2.1: QRD RLS systolic array

(1) Boundary Cell



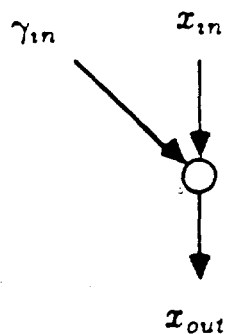
If $x_{in} = 0$ **then**
 $c \leftarrow 1; s \leftarrow 0; \gamma_{out} \leftarrow \gamma_{in};$
 $r = \lambda r,$
otherwise
 $r' = \sqrt{\lambda^2 r^2 + x_{in}^2};$
 $c \leftarrow \lambda r / r'; s \leftarrow x_{in} / r';$
 $r \leftarrow r'; \gamma_{out} = c \gamma_{in}$
end

(2) Internal Cell



$x_{out} \leftarrow c x_{in} - s \lambda r$
 $r \leftarrow s x_{in} + c \lambda r$

(3) Final Cell



$x_{out} \leftarrow \gamma_{in} x_{in}$

Figure 2.2: Processing Cells of QRD RLS systolic array

2.2 Systolic Block Householder Transformation

The Givens rotation method discussed above is a rank-1 update approach since each input is a row of data. For the systolic block Householder transformation (SBHT), we need a block data formulation. Denote the *data matrix* as

$$\mathbf{X}(n) = \begin{bmatrix} \mathbf{X}_1^T \\ \mathbf{X}_2^T \\ \vdots \\ \mathbf{X}_n^T \end{bmatrix} = \begin{bmatrix} \mathbf{X}(n-1) \\ - - - \\ \mathbf{X}_n^T \end{bmatrix} \in \mathbb{R}^{nk \times p} \quad (2.6)$$

and the *desired response vector* as

$$\mathbf{y}(n) = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{bmatrix} = \begin{bmatrix} \mathbf{y}(n-1) \\ - - - \\ \mathbf{y}_n \end{bmatrix} \in \mathbb{R}^{nk}, \quad (2.7)$$

where \mathbf{X}_i^T is the i^{th} data block,

$$\mathbf{X}_i^T = \begin{bmatrix} \mathbf{x}_{(i-1)k+1}^T \\ \mathbf{x}_{(i-1)k+2}^T \\ \vdots \\ \mathbf{x}_{ik}^T \end{bmatrix} = [\mathbf{x}_{i,1} \ \mathbf{x}_{i,2} \ \cdots \ \mathbf{x}_{i,p}] \quad (2.8)$$

$$= \begin{bmatrix} x_{(i-1)k+1,1} & x_{(i-1)k+1,2} & \cdots & x_{(i-1)k+1,p} \\ x_{(i-1)k+2,1} & x_{(i-1)k+2,2} & \cdots & x_{(i-1)k+2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{ik,1} & x_{ik,2} & \cdots & x_{ik,p} \end{bmatrix} \in \mathbb{R}^{k \times p} \quad (2.9)$$

and \mathbf{y}_i is the i^{th} desired response block,

$$\mathbf{y}_i = \begin{bmatrix} y_{(i-1)k+1} \\ y_{(i-1)k+2} \\ \vdots \\ y_{ik} \end{bmatrix} \in \mathbb{R}^k. \quad (2.10)$$

k is the *block size* and p is the *order (columns)* of the system.

For a rank- k update QR decomposition, suppose we have

$$\mathbf{Q}(n-1)\mathbf{X}(n-1) = \begin{bmatrix} \mathbf{R}(n-1) \\ - - - \\ 0 \end{bmatrix}. \quad (2.11)$$

Denote

$$\bar{\mathbf{Q}}_k(n-1) = \begin{bmatrix} \mathbf{Q}(n-1) & \vdots & \mathbf{0} \\ - - - & & - - - \\ \mathbf{0}^T & \vdots & \mathbf{I}_k \end{bmatrix}, \quad (2.12)$$

then we have

$$\bar{\mathbf{Q}}_k(n-1)\mathbf{X}(n) = \begin{bmatrix} \mathbf{R}(n-1) \\ - - - \\ \mathbf{0} \\ - - - \\ \mathbf{X}_n^T \end{bmatrix}. \quad (2.13)$$

If we can find a matrix $\mathbf{H}(n)$ such that

$$\mathbf{H}(n)\bar{\mathbf{Q}}_k(n-1)\mathbf{X}(n) = \begin{bmatrix} \mathbf{R}(n) \\ - - - \\ 0 \end{bmatrix}, \quad (2.14)$$

then the new $\mathbf{Q}(n)$ is

$$\mathbf{Q}(n) = \mathbf{H}(n)\bar{\mathbf{Q}}_k(n-1). \quad (2.15)$$

An $n \times n$ Householder transformation matrix \mathbf{T} is of the form

$$\mathbf{T} = \mathbf{I} - \frac{2\mathbf{z}\mathbf{z}^T}{\|\mathbf{z}\|^2}, \quad (2.16)$$

where $\mathbf{z} \in \mathbb{R}^n$. When a vector \mathbf{x} is multiplied by \mathbf{T} , it is reflected in the hyperplane defined by $\text{span}\{\mathbf{z}\}^\perp$. Choosing $\mathbf{z} = \mathbf{x} \pm \|\mathbf{x}\|_2 \mathbf{e}_1$, where $\mathbf{e}_1 = [1, 0, 0, \dots, 0] \in \mathbb{R}^n$, then \mathbf{x} is reflected onto \mathbf{e}_1 by \mathbf{T} as

$$\mathbf{T}\mathbf{x} = \pm \|\mathbf{x}\|_2 \mathbf{e}_1. \quad (2.17)$$

That is, all of the energy of \mathbf{x} is reflected onto unit vector \mathbf{e}_1 after the transformation.

We can zero out \mathbf{X}_n^T by applying successive Householder transformations as follows,

$$\mathbf{H}^{(i)}(n) \begin{bmatrix} \mathbf{R}^{(i-1)}(n-1) \\ \text{---} \\ \mathbf{0} \\ \text{---} \\ 0, \dots, 0, \mathbf{x}_{n,i}^{(i-1)}, \dots, \mathbf{x}_{n,p}^{(i-1)} \end{bmatrix} = \begin{bmatrix} \mathbf{R}^{(i)}(n-1) \\ \text{---} \\ \mathbf{0} \\ \text{---} \\ 0, \dots, 0, 0, \mathbf{x}_{n,i+1}^{(i)}, \dots, \mathbf{x}_{n,p}^{(i)} \end{bmatrix},$$

for $i = 1, \dots, p$, where $\mathbf{x}_{n,i}^{(0)} = \mathbf{x}_{n,i}$, $\mathbf{R}^{(0)}(n-1) = \mathbf{R}(n-1)$, and the resultant matrix $\mathbf{H}(n)$ is

$$\mathbf{H}(n) = \mathbf{H}^{(p)}(n) \mathbf{H}^{(p-1)}(n) \dots \mathbf{H}^{(1)}(n), \quad (2.18)$$

where each $\mathbf{H}^{(i)}(n)$ represents a Householder transformation which zeros out the i^{th} column of the updated \mathbf{X}_n^T , i.e., $\mathbf{x}_{n,i}^{(i-1)}$.

To obtain $\mathbf{H}^{(1)}(n)$, denote

$$\mathbf{z}_1 = \begin{bmatrix} r_{11} - \sigma_1 & \vdots & \mathbf{0}_{(n-1)k-1}^T & \vdots & \mathbf{x}_{n,1}^T \end{bmatrix}^T,$$

where r_{11} is the $(1,1)$ element of $\mathbf{R}(n-1)$, $\sigma_1^2 = r_{11}^2 + \|\mathbf{x}_{n,1}\|^2$. Then from (2.16)

$$\mathbf{H}^{(1)}(n) = \begin{bmatrix} h_{11}^{(1)}(n) & \vdots & \mathbf{0}^T & \vdots & \mathbf{h}_{12}^{(1)T}(n) \\ \text{---} & & \text{---} & & \text{---} \\ \mathbf{0} & \vdots & I_{(n-1)k-1} & \vdots & \mathbf{0} \\ \text{---} & & \text{---} & & \text{---} \\ \mathbf{h}_{21}^{(1)}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{22}^{(1)}(n) \end{bmatrix}, \quad (2.19)$$

where $h_{11}^{(1)}(n)$ is a scalar, $\mathbf{h}_{12}^{(1)}(n)$ is a $k \times 1$ vector, $\mathbf{h}_{21}^{(1)}(n) = \mathbf{h}_{12}^{(1)}(n)$, and $\mathbf{H}_{22}^{(1)}(n)$ is a $k \times k$ matrix and

$$\mathbf{H}_{22}^{(1)}(n) = \mathbf{I}_k - \frac{2\mathbf{x}_{n,1}\mathbf{x}_{n,1}^T}{\sigma_{\mathbf{z}_1}^2}, \quad (2.20)$$

with $\sigma_{\mathbf{z}_1}^2 = \|\mathbf{z}_1\|_2^2 = 2(\sigma_1^2 - \sigma_1 r_{11})$. Define $\psi_1 = \sigma_1^2 - \sigma_1 r_{11}$, (2.20) can be rewritten in a form without multiplication of the number 2 as

$$\mathbf{H}_{22}^{(1)} = \mathbf{I} - \frac{\mathbf{x}_{n,1}\mathbf{x}_{n,1}^T}{\psi_1}.$$

In general,

$$\mathbf{H}^{(m)}(n) = \begin{bmatrix} \mathbf{H}_{11}^{(m)}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{12}^{(m)}(n) \\ \dots & . & \dots & . & \dots \\ \mathbf{0} & \vdots & \mathbf{I}_{(n-1)k-p} & \vdots & \mathbf{0} \\ \dots & . & \dots & . & \dots \\ \mathbf{H}_{21}^{(m)}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{22}^{(m)}(n) \end{bmatrix}, \quad (2.21)$$

where $\mathbf{H}_{11}^{(m)}(n) \in \mathbb{R}^{p \times p}$ is an identity matrix except for the m^{th} diagonal entry; $\mathbf{H}_{12}^{(m)}(n) \in \mathbb{R}^{p \times k}$ is a zero matrix except for the m^{th} row; $\mathbf{H}_{21}^{(m)}(n) = \mathbf{H}_{12}^{(m)T}(n)$; and

$$\mathbf{H}_{22}^{(m)}(n) = \mathbf{I}_k - \frac{\mathbf{x}_{n,m}^{(m-1)} \mathbf{x}_{n,m}^{(m-1)T}}{\psi_m} \in \mathbb{R}^{k \times k} \quad (2.22)$$

is symmetric with $\psi_m = \sigma_m^2 - \sigma_m r_{mm}$, where $\sigma_m^2 = r_{mm}^2 + \|\mathbf{x}_{n,m}^{(m-1)}\|^2$. It can be easily seen that $\mathbf{H}_{12}^{(i)}(n) \mathbf{H}_{21}^{(j)}(n) = \mathbf{0}$, $\mathbf{H}_{21}^{(i)}(n) \mathbf{H}_{12}^{(j)}(n) = \mathbf{0}$, for $\forall i \neq j$. Thus we have the following lemma,

Lemma 2.1 *The Householder transformation matrix, $\mathbf{H}(n) \in \mathbb{R}^{nk \times nk}$, is orthogonal and is of the form*

$$\mathbf{H}(n) = \begin{bmatrix} \mathbf{H}_{11}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{12}(n) \\ \dots & . & \dots & . & \dots \\ \mathbf{0} & \vdots & \mathbf{I}_{(n-1)k-p} & \vdots & \mathbf{0} \\ \dots & . & \dots & . & \dots \\ \mathbf{H}_{21}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{22}(n) \end{bmatrix} = \mathbf{H}^{(p)}(n) \mathbf{H}^{(p-1)}(n) \dots \mathbf{H}^{(1)}(n), \quad (2.23)$$

with

$$\begin{aligned} \mathbf{H}_{11}(n) &= \mathbf{H}_{11}^{(p)}(n) \dots \mathbf{H}_{11}^{(2)}(n) \mathbf{H}_{11}^{(1)}(n) \\ \mathbf{H}_{22}(n) &= \mathbf{H}_{22}^{(p)}(n) \dots \mathbf{H}_{22}^{(2)}(n) \mathbf{H}_{22}^{(1)}(n). \square \end{aligned} \quad (2.24)$$

If the block size $k = 1$, due to the fact that Givens rotation method is a special case of rank-1 update Householder transformation [30], the \mathbf{H} matrix in *Lemma 2.1* becomes a Givens rotation matrix \mathbf{G} of the form [78]

$$\mathbf{G}(n) = \left[\begin{array}{c|c|c} \mathbf{K}(n) & \mathbf{0} & \mathbf{h}(n) \\ \hline \mathbf{0} & \mathbf{I}_{n-p-1} & \mathbf{0} \\ \hline \mathbf{h}^H(n) & \mathbf{0} & \gamma(n) \end{array} \right]$$

where $\mathbf{K}(n)$ is a $p \times p$ matrix, $\mathbf{h}(n)$ is a $p \times 1$ vector, and $\gamma(n)$ is a scalar given by $\gamma(n) = \prod_{i=1}^p c_i(n)$, $n \geq p$ where $c_i(n)$ is the cosine parameter associated with the i^{th} Given rotation.

2.2.1 Vectorized SBHT QRD Systolic Array

Now we propose a vectorized systolic array to implement the QRD based on the SBHT. Similar to the QR triarray of Gentleman-Kung [29], this array has both boundary and internal cells. The boundary cell takes an input of block size k from the previous processor or directly from the input port, updates its content and generates reflection vector, and sends it to the right for the internal cell processing. Define

$$\bar{\mathbf{x}}_{n,i}^{(i-1)T} = \left[\mathbf{0}_{i-1} \quad \vdots \quad r_{ii} \quad \vdots \quad \mathbf{0}_{(n-1)k-i} \quad \vdots \quad \mathbf{x}_{n,i}^{(i-1)T} \right], \quad i = 1, \dots, p,$$

and $\mathbf{z}_i = \bar{\mathbf{x}}_{n,i}^{(i-1)} - \sigma_i \mathbf{e}_i$, where \mathbf{e}_i is a zero vector except for a unity at the i^{th} position. When an internal cell receives the reflection vector, instead of forming the matrix $\mathbf{z}_i \mathbf{z}_i^T$ and performing matrix arithmetics, it performs an inner product operation to update its content r_{ij} by doing

$$\mathbf{H}^{(i)}(n) \bar{\mathbf{x}}_{n,j}^{(i-1)} = \bar{\mathbf{x}}_{n,j}^{(i-1)} - \frac{\mathbf{z}_i}{\psi_i} (\mathbf{z}_i^T \cdot \bar{\mathbf{x}}_{n,j}^{(i-1)}), \quad j = i + 1, \dots, p, \quad (2.25)$$

and sends the reflected data $\mathbf{x}_{n,j}$ downward for further processing. Fig. 2.3 and Fig. 2.4 show the SBHT QRD array architecture and the operations of the processing cells respectively. When the block size is $k = 1$, this vectorized array degenerates to the Gentleman-Kung's Givens rotation triarray.

2.3 SBHT RLS Algorithm

The LS problem is to choose a *weight vector* $\mathbf{w}(n) \in \mathbb{R}^p$, such that the *block-forgetting norm* of

$$\underline{\mathbf{e}}(n) = \begin{bmatrix} \mathbf{e}_1(n) \\ \mathbf{e}_2(n) \\ \vdots \\ \mathbf{e}_n(n) \end{bmatrix} = \mathbf{X}(n)\mathbf{w}(n) - \mathbf{y}(n) \quad (2.26)$$

is minimized. That is,

$$\hat{\mathbf{w}}(n) = \arg \min_{\mathbf{w}} \|\underline{\mathbf{e}}(n)\|_{\Lambda_k}, \quad (2.27)$$

where

$$\|\underline{\mathbf{e}}(n)\|_{\Lambda_k} = \|\Lambda_k(n)\underline{\mathbf{e}}(n)\| = \sqrt{\sum_{i=1}^n \lambda^{2(n-i)} \cdot \|\mathbf{e}_i(n)\|_2^2}, \quad 0 < \lambda \leq 1, \quad (2.28)$$

$\Lambda_k(n)$ is a block-diagonal exponential weighting matrix of the form,

$$\Lambda_k(n) = \begin{bmatrix} \lambda^{n-1}\mathbf{I}_k & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \cdots & \lambda\mathbf{I}_k & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I}_k \end{bmatrix} \in \mathbb{R}^{nk \times nk}, \quad (2.29)$$

and $\|\cdot\|_2$ is the Euclidean norm,

$$\|\mathbf{e}_i(n)\|_2^2 = \sum_{j=1}^k |e_{(i-1)k+j}(n)|^2. \quad (2.30)$$

The exponential forgetting weighting λ is incorporated in the RLS filtering scheme to avoid overflow in the processors as well as to facilitate nonstationary data updating.

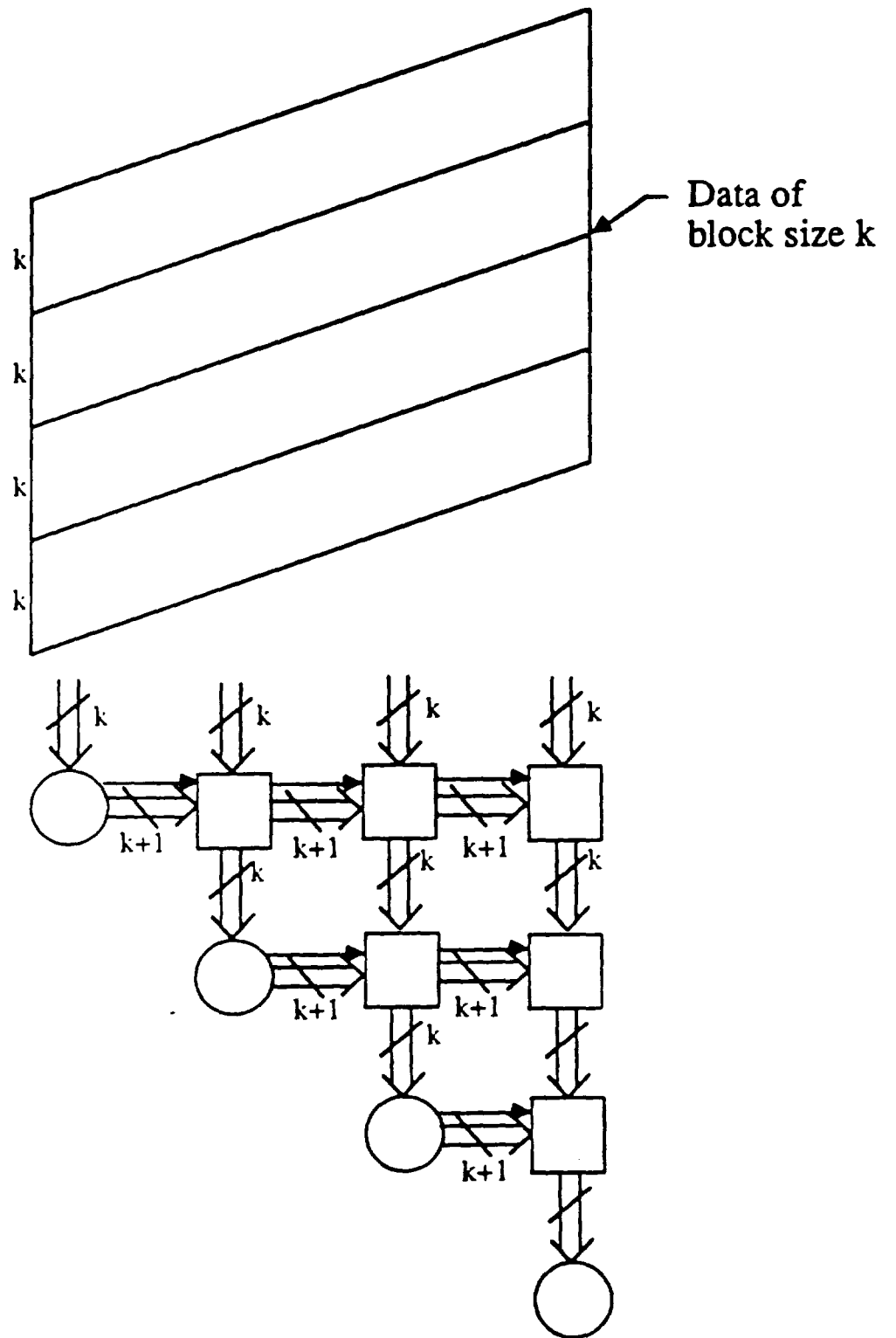
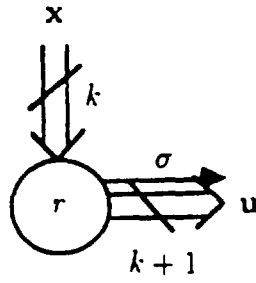
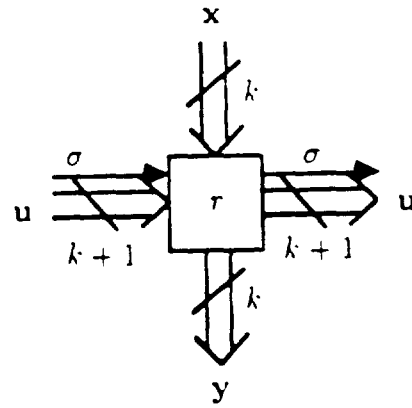


Figure 2.3: The SBHT QRD systolic array.



If $\|x\|^2 = 0$ then
 $\sigma = 0, r = \lambda r,$
else
 $S \leftarrow \lambda^2 r^2 + \|x\|_2^2$
 $s \leftarrow \sqrt{S}$
 $\sigma \leftarrow S + s\lambda r$
 $u \leftarrow \begin{bmatrix} \lambda r + s \\ x \end{bmatrix}$
 $r \leftarrow s$
end



If $\sigma = 0$ then
 $y = x, r = \lambda r.$
else
 $t \leftarrow \sigma^{-1} \cdot u^T \cdot \begin{bmatrix} \lambda r \\ x \end{bmatrix}$
 $\begin{bmatrix} r \\ y \end{bmatrix} \leftarrow \begin{bmatrix} \lambda r \\ x \end{bmatrix} - t \cdot u$
end

Figure 2.4: The processing cells of the SBHT QRD systolic array.

If we know the QRD of the *weighted augmented data matrix* at time n (in the block sense, which is equivalent to the nk snapshots), which is given by

$$\Lambda_k(n) \begin{bmatrix} \mathbf{X}(n) \\ \mathbf{y}(n) \end{bmatrix} = [\mathbf{Q}_1^T(n) : \mathbf{Q}_2^T(n)] \begin{bmatrix} \mathbf{R}(n) & \vdots & \mathbf{u}(n) \\ \mathbf{0} & \vdots & \mathbf{v}(n) \end{bmatrix}, \quad (2.31)$$

where

$$\mathbf{Q}(n) = \begin{bmatrix} \mathbf{Q}_1(n) \\ \hline \mathbf{Q}_2(n) \end{bmatrix},$$

and $\mathbf{Q}_1(n) \in \mathbb{R}^{p \times nk}$ and $\mathbf{Q}_2(n) \in \mathbb{R}^{(nk-p) \times nk}$ constitute an orthogonal transformation matrix with the former spanning the column space of the weighted data matrix $\Lambda_k(n)\mathbf{X}(n)$ and the latter the null space, and $\mathbf{R}(n) \in \mathbb{R}^{p \times p}$ is an upper triangular matrix. The optimal weight vector can be obtained by solving

$$\mathbf{R}(n)\hat{\mathbf{w}}(n) = \mathbf{u}(n). \quad (2.32)$$

Obviously, $\Lambda_k(n)\mathbf{X}(n) = \mathbf{Q}_1^T(n)\mathbf{R}(n)$. As a result, the weighted optimal residual of (2.26) is,

$$\begin{aligned} \Lambda_k(n)\hat{\mathbf{e}}(n) &= \mathbf{Q}_1^T(n)\mathbf{R}(n)\hat{\mathbf{w}}(n) - \mathbf{Q}_1^T(n)\mathbf{u}(n) - \mathbf{Q}_2^T(n)\mathbf{v}(n) \\ &= -\mathbf{Q}_2^T(n)\mathbf{v}(n), \end{aligned} \quad (2.33)$$

which lies in the null space of the weighted data matrix.

Now, suppose we have the data matrix up to time $n-1$ and the QRD of $\Lambda_k(n-1)[\mathbf{X}(n-1) : \mathbf{y}(n-1)]$, then the recursive LS problem is to efficiently compute the optimum residual at time n from the results we have at time $n-1$. In particular, we are interested in the new n^{th} block of the optimal residual,

$$\hat{\mathbf{e}}_n(n) = \mathbf{X}_n^T \hat{\mathbf{w}}(n) - \mathbf{y}_n. \quad (2.34)$$

From (2.13), (2.14) and (2.23), (2.31) can be expressed as

$$\begin{bmatrix} \mathbf{R}(n) & \vdots & \mathbf{u}(n) \\ \mathbf{0} & \vdots & \mathbf{v}(n) \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{11}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{12}(n) \\ \dots & \vdots & \dots & \vdots & \dots \\ \mathbf{0} & \vdots & \mathbf{I}_{(n-1)k-p} & \vdots & \mathbf{0} \\ \dots & \vdots & \dots & \vdots & \dots \\ \mathbf{H}_{21}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{22}(n) \end{bmatrix} \cdot \begin{bmatrix} \lambda \mathbf{R}(n-1) & \vdots & \lambda \mathbf{u}(n-1) \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{0} & \vdots & \lambda \mathbf{v}(n-1) \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{X}_n^T & \vdots & \mathbf{y}_n \end{bmatrix}.$$

By recursion on n , we relate $\mathbf{Q}(n)$ and $\mathbf{Q}(n-1)$ using (2.15) and have

$$\begin{aligned} \mathbf{Q}(n) &= \begin{bmatrix} \mathbf{H}_{11}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{12}(n) \\ \dots & \vdots & \dots & \vdots & \dots \\ \mathbf{0} & \vdots & \mathbf{I}_{(n-1)k-p} & \vdots & \mathbf{0} \\ \dots & \vdots & \dots & \vdots & \dots \\ \mathbf{H}_{21}(n) & \vdots & \mathbf{0} & \vdots & \mathbf{H}_{22}(n) \end{bmatrix} \cdot \begin{bmatrix} \mathbf{Q}_1(n-1) & \vdots & \mathbf{0} \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{Q}_2(n-1) & \vdots & \mathbf{0} \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{0} & \vdots & \mathbf{I}_k \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{H}_{11}(n)\mathbf{Q}_1(n-1) & \vdots & \mathbf{H}_{12}(n) \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{Q}_2(n-1) & \vdots & \mathbf{0} \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{H}_{21}(n)\mathbf{Q}_1(n-1) & \vdots & \mathbf{H}_{22}(n) \end{bmatrix}. \end{aligned} \quad (2.35)$$

We can see that $\mathbf{Q}_2(n)$ is updated from $\mathbf{Q}_1(n-1)$ and $\mathbf{Q}_2(n-1)$ by

$$\mathbf{Q}_2(n) = \begin{bmatrix} \mathbf{Q}_2(n-1) & \vdots & \mathbf{0} \\ \text{-----} & \vdots & \text{-----} \\ \mathbf{H}_{21}(n)\mathbf{Q}_1(n-1) & \vdots & \mathbf{H}_{22}(n) \end{bmatrix}. \quad (2.36)$$

On the other hand, the updated $[\mathbf{u}^T(n), \mathbf{v}^T(n)]^T$ is

$$\begin{aligned} \begin{bmatrix} \mathbf{u}(n) \\ \text{-----} \\ \mathbf{v}(n) \end{bmatrix} &= \mathbf{Q}(n)\mathbf{\Lambda}_k(n) \begin{bmatrix} \mathbf{y}(n-1) \\ \text{-----} \\ \mathbf{y}_n \end{bmatrix} \\ &= \begin{bmatrix} \lambda \mathbf{H}_{11}(n)\mathbf{u}(n-1) + \mathbf{H}_{12}(n)\mathbf{y}_n \\ \text{-----} \\ \lambda \mathbf{v}(n-1) \\ \text{-----} \\ \mathbf{v}_n \end{bmatrix}, \end{aligned} \quad (2.37)$$

where

$$\mathbf{v}_n = \lambda \mathbf{H}_{21}(n)\mathbf{u}(n-1) + \mathbf{H}_{22}(n)\mathbf{y}_n. \quad (2.38)$$

Therefore, from (2.33), (2.36), and (2.37), the weighted optimal residual vector can be obtained from parameters in time $n - 1$ by

$$\Lambda_k(n)\hat{\underline{e}}(n) = \begin{bmatrix} \hat{\underline{e}}(n-1|n) \\ \vdots \\ \hat{\mathbf{e}}_n(n) \end{bmatrix} = \begin{bmatrix} -\lambda \mathbf{Q}_2^T(n-1)\mathbf{v}(n-1) - \mathbf{Q}_1^T(n-1)\mathbf{H}_{21}^T(n)\mathbf{v}_n \\ \vdots \\ -\mathbf{H}_{22}^T(n)\mathbf{v}_n \end{bmatrix}, \quad (2.39)$$

where $\hat{\underline{e}}(m|n)$ denotes the estimate of $\hat{\underline{e}}$ at time m , $m \leq n$, given all of the data up to time n . The new n^{th} block of the optimal residual is then obtained as

$$\hat{\mathbf{e}}_n(n) = -\mathbf{H}_{22}^T(n)\mathbf{v}_n = -\mathbf{H}_{22}^{(1)}(n)\mathbf{H}_{22}^{(2)}(n) \cdots \mathbf{H}_{22}^{(p)}(n)\mathbf{v}_n. \quad (2.40)$$

For the block size of $k = 1$, all vector parameters in (2.40) become scalars and can be expressed as

$$e_n(n) = -\prod_{i=1}^p c_i v_n, \quad (2.41)$$

which was first shown by McWhirter in [78]. Note that there is little difference in the optimal residuals estimated by the SBHT and the Givens rotation methods. To be specific, the optimal residual vector in (2.40) is given by

$$\hat{\mathbf{e}}_n(n) = \begin{bmatrix} e_{(n-1)k+1}((n-1)k+1|nk) \\ \vdots \\ e_{nk-1}(nk-1|nk) \\ e_{nk}(nk|nk) \end{bmatrix}, \quad (2.42)$$

while the optimal residual estimated by the Givens rotation method in (2.41) is

$$\hat{e}_n(n) = e_n(n|n). \quad (2.43)$$

In this sense, the SBHT RLS gives a better estimate of the residual since it uses more data samples to estimate the optimal residual. Take $k = 2$ as an example;

the optimal residual obtained from the SBHT RLS and the Givens methods are $[e_{2n-1}((2n-1)|2n), e_{2n}(2n|2n)]$ and $[e_{2n-1}((2n-1)|(2n-1), e_{2n}(2n|2n)]$ respectively. It is clear now that the SBHT RLS method gives a better estimate for the previous residual than the Givens rotation method because the former makes use of the future data sample at time $2n$ to estimate the residual at time $2n-1$, while the latter does not.

2.3.1 Vectorized SBHT RLS Array

To obtain the residual vector for RLS filtering in the systolic array, there are two possible approaches. The first one is to generalize the architecture of McWhirter's Givens rotation approach [78]. A SBHT QRD array is incorporated with a RA as shown in Fig. 2.5. Observe that \mathbf{v}_n in (2.38) results from the reflection computation in (2.37), therefore \mathbf{v}_n is obtained naturally from the output of the RA. Each boundary cell then forms the matrix $\mathbf{H}_{22}^{(i)}$ and propagates it down diagonal boundary cells. Since $\mathbf{H}_{22}^{(i)}$ is generated earlier than $\mathbf{H}_{22}^{(j)}$ for $i < j$, equation (2.30) has to be computed from left to right. Obviously, we prefer to compute (2.40) from right to left such that only inner product computations are performed instead of matrix multiplications. As a result, each boundary cell performs the matrix multiplication to cumulate $\mathbf{H}_{22}^{(i)}$ when it is propagated down diagonal boundary cells. The matrix multiplications needed in the boundary cells in this approach are objectionable since they not only slow down the throughput but also increase the complexity of the boundary cells. We note, McWhirter's original approach based on Givens rotation worked well since only scalars need to be propagated down the diagonal boundary cells and the multiplications for scalars can be in the reverse order.

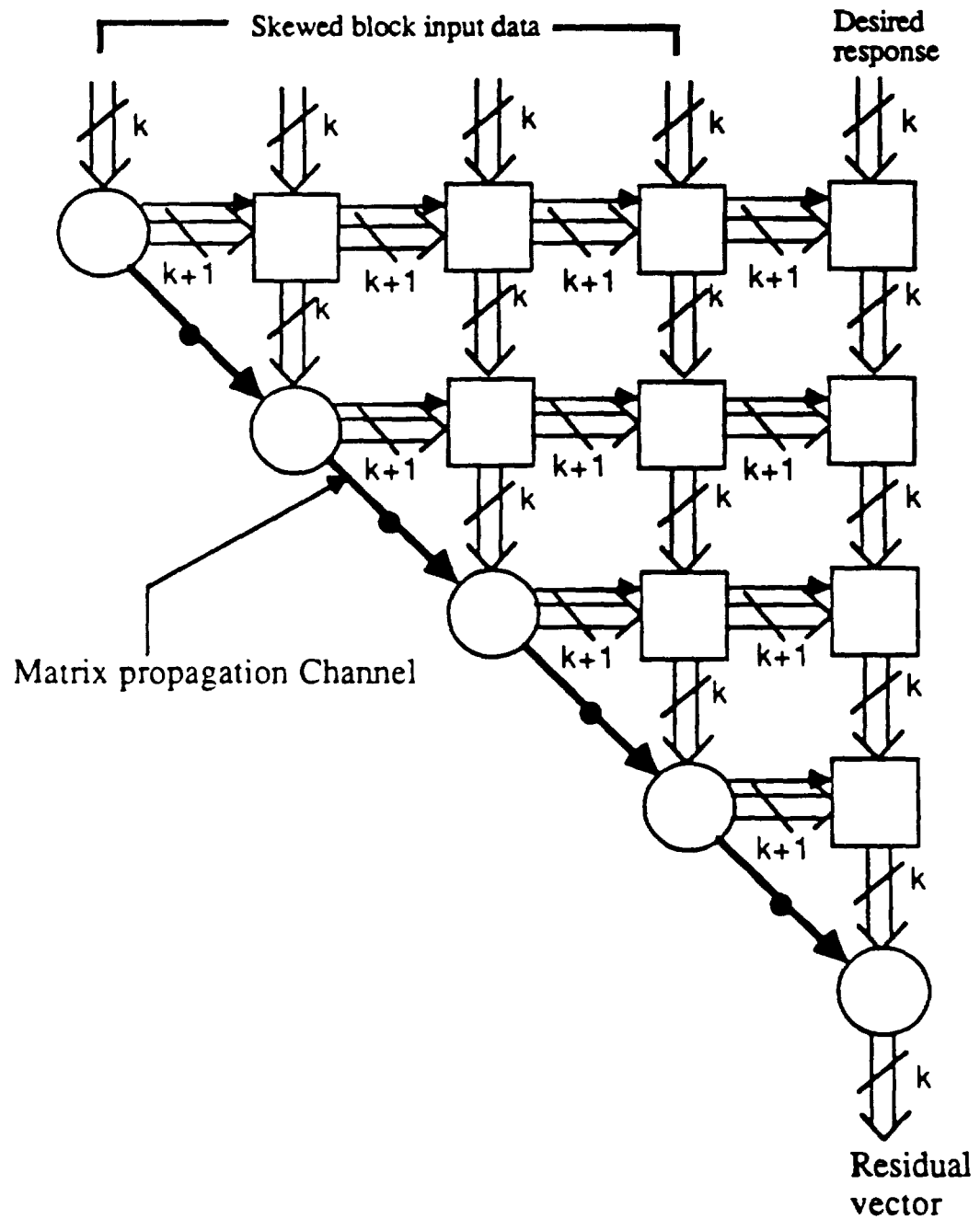


Figure 2.5: The SBHT RLS systolic array obtained by direct generalization of the Givens rotation array.

Instead of forming the matrix $\mathbf{H}_{22}^{(i)}$ and propagating it down, another approach is to use the facts that $\mathbf{H}_{22}^{(i)}$ can be expressed by using (2.22) and the reflection vectors are sent to the right from boundary cells as described in Section 2.2.1. From these observations, (2.40) can be computed in similar operations performed by the internal cells. A generalized architecture, as shown in Fig. 2.6, is thus introduced to circumvent this problem. A column array of internal cells called *backward propagation array* (BPA) is added at the right hand side to perform the *backward propagation* of \mathbf{v}_n . Each row, say the i^{th} one, needs $2(p - i)$ delayed buffers as shown in Fig. 2.6. The \mathbf{v}_n obtained at the output of RA is then backward propagated through the BPA. From (2.22), each cell of this array performs the operation

$$\mathbf{H}_{22}^{(i)}(n)\mathbf{v}'_n = \mathbf{v}'_n - \frac{\mathbf{x}_{n,i}^{(i-1)}}{\psi_i}(\mathbf{x}_{n,i}^{(i-1)T} \cdot \mathbf{v}'_n), \quad i = p, \dots, 2, 1, \quad (2.44)$$

where \mathbf{v}'_n is an updated \mathbf{v}_n . This is a subset of the operations performed by the internal cell shown in (2.25). The residual vector is obtained from the top of the newly added column array.

The costs for this proposed architecture are: an increased latency time from $(2p + 1)t_s$ of McWhirter's Givens method to $3pt_v$, where t_s represents the processing time for the scalar operations used in the Givens rotation method and t_v is the processing time for vector operations used in the SBHT method; the number of delay elements needed increases from p to $\sum_1^p 2(p - i) = p(p - 1)$; and p additional internal processing cells. The operations of the boundary and internal cells are given in Fig. 2.5. These results clearly show that HT can be implemented simply on a systolic array to achieve massive parallel processing with vector operations. This provides an efficient method to obtain a high throughput rate for recursive LS filtering by using the HT method.

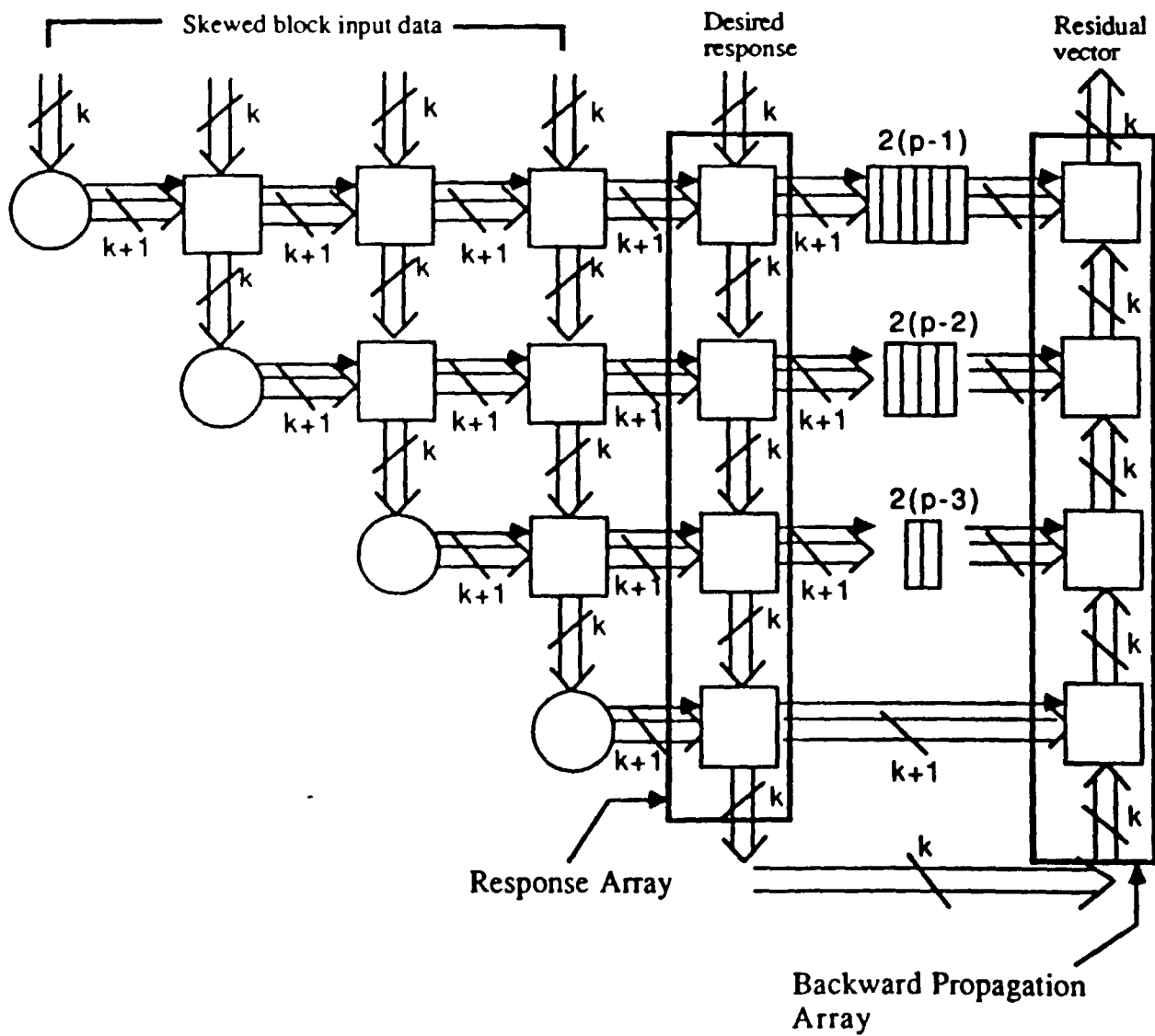


Figure 2.6: The SBHT RLS systolic array.

2.4 Two-level Pipelined Implementations

The SBHT QRD array and RLS array discussed in the above sections are derived from the conventional Householder transformation shown in (2.16). Due to the vector processing nature of the conventional method, the cells of both arrays perform vector operations such as inner products. This means the complexity of each cell is high and the I/O bandwidth is large in order to have an effective communication of vector data. Each cell, due to the complexity for vector processing, may lead to a large processor. Clearly, this is not a desirable property for VLSI implementation. Thus, we are motivated to find a suitable algorithm to pipeline the data down to the word level such that the I/O bandwidth as well as the complexity can be reduced. In addition, we still wish to achieve a high throughput which is needed in many modern signal processing applications.

The conventional approach in computing Householder transformation, $\mathbf{y} = \mathbf{T}\mathbf{q}$, based on (2.16) is to form \mathbf{z} and $\|\mathbf{z}\|^2$ from \mathbf{x} first and then $\mathbf{z}^T\mathbf{q}/\|\mathbf{z}\|^2$ and $\mathbf{q} - 2\mathbf{z}(\mathbf{z}^T\mathbf{q}/\|\mathbf{z}\|^2)$ as considered before. It can be stated in the following form:

HT Algorithm (Conventional)

Step 1. $S_{xx} = \|\mathbf{x}\|^2$.

Step 2. If $S_{xx} = 0$, then $\mathbf{y} = \mathbf{q}$.

Step 3. If $S_{xx} \neq 0$ then

(1) $s = \sqrt{S_{xx}}$, $\mathbf{z} = \mathbf{x} + [s, 0, 0, \dots, 0]^T$,

(2) $\phi = S_{xx} + sx_1$, $S_{zq} = \mathbf{z}^T\mathbf{q}$,

(3) $d = S_{zq}/\phi$, $\mathbf{y} = \mathbf{q} - d\mathbf{z}$.

In [104], Tsao pointed out that by skipping the computation of ϕ and avoiding the cumbersome intermediate steps of forming vector \mathbf{z} for further computations, a modified algorithm with smaller round off error and less operations can be obtained.

Only step 3 of the conventional algorithm is to be modified as follows,

Modified HT Algorithm (Tsao, 1975)

Step 3. If $S_{xx} \neq 0$ then

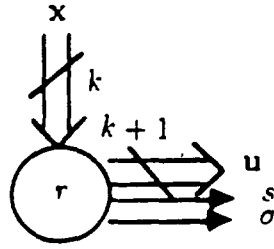
$$(1) s = \sqrt{S_{xx}}, \quad \sigma = x_1 + s,$$

$$(2) S_{xq} = \mathbf{x}^T \mathbf{q},$$

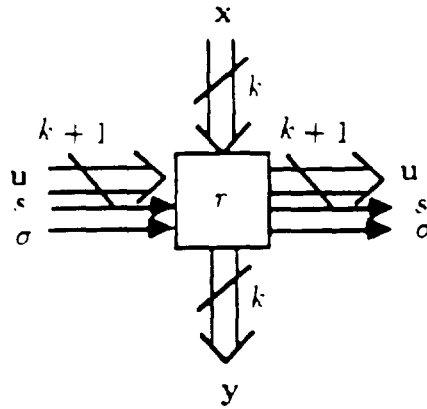
$$(3) y_1 = -S_{xq}/s, \quad d = (q_1 - y_1)/\sigma, \quad y_i = q_i - dx_i, \quad i = 2, \dots, n.$$

With this algorithm, the operations of the cells of the vectorized systolic arrays can be modified as shown in Fig. 2.7. As we can see that, for the boundary cell, the vector \mathbf{u} , which consists of the forgotten diagonal element of the upper-triangular matrix and one column of the input data block (updated or not), can be sent out immediately when the input \mathbf{x} is available without waiting for any computations as required in the implementation using the conventional algorithm. Due to this advantage and the modified operations in the internal cells, we can then pipeline the vector operations down to the word level such that each cell only performs scalar operations which will significantly reduce the complexity of the cell.

A two-level pipelined implementation of the modified HT algorithm is given in Fig. 2.8 and Fig. 2.9. The boundary cell performs three major functions - square-and-cumulate, square root, and addition. For each data block, the boundary cell fetches one data sample, cumulates the square of the sample, and sends the data to the right for internal cell processing. When all the data of the block are processed, the content of S is then sent down for square-root operation. The resultant s is sent to the right for internal cell processing as well as sent down to obtain σ , which is then sent to the right when available. At the same time, when an internal cell receives a u_i , it multiplies u_i with an input x_i and cumulates all these products to obtain S . When S is available, it is sent down for division operation with s , which arrives at



If $\|x\|^2 = 0$ **then**
 $\sigma = 0, r = \lambda r,$
else
 $S \leftarrow \lambda^2 r^2 + \|x\|_2^2$
 $s \leftarrow \sqrt{S}$
 $\sigma \leftarrow s + \lambda r$
 $u \leftarrow \begin{bmatrix} \lambda r \\ x \end{bmatrix}$
 $r \leftarrow s$
end



If $\sigma = 0$ **then**
 $y = x, r = \lambda r,$
else
 $S \leftarrow u^T \cdot \begin{bmatrix} \lambda r \\ x \end{bmatrix}$
 $t \leftarrow -S/s$
 $d \leftarrow (\lambda r - t)/\sigma$
 $r \leftarrow t$
 $y \leftarrow x - d \cdot \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{k+1} \end{bmatrix}$
end

Figure 2.7: Operations of the processing cells by using modified Householder transformation.

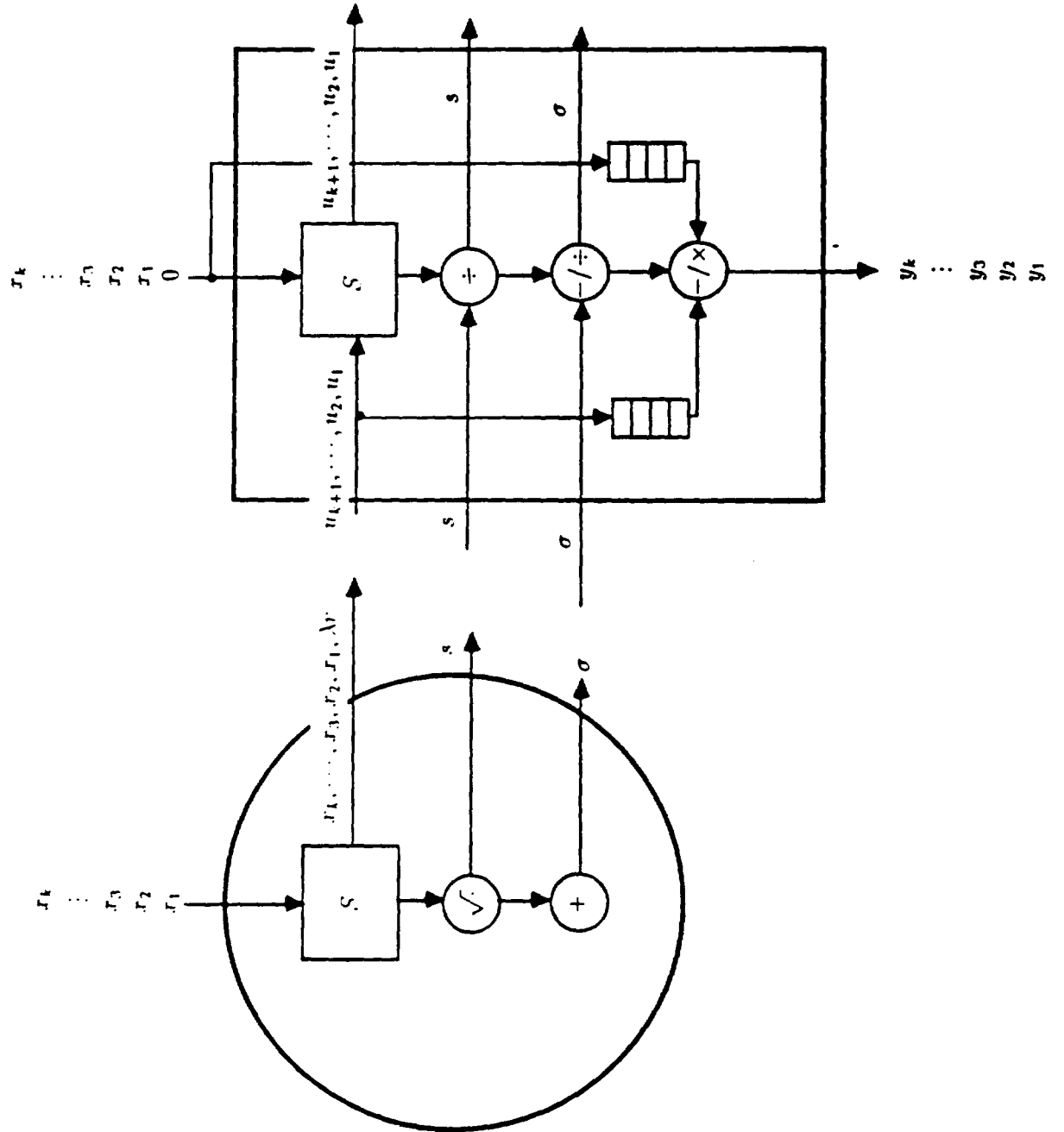
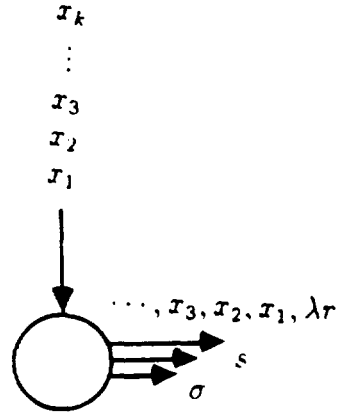


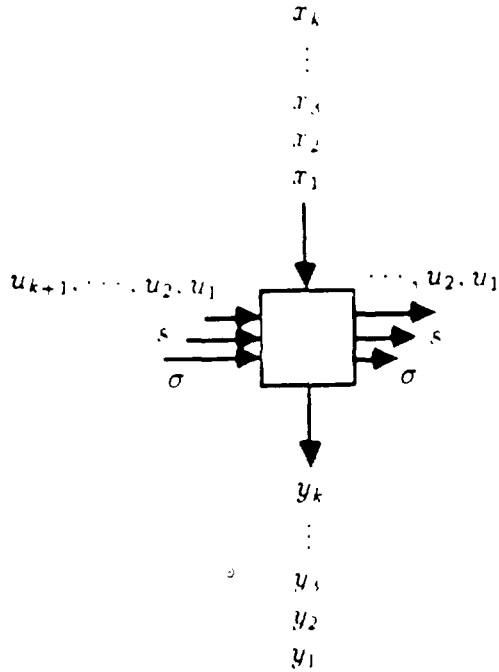
Figure 2.8: Processing cells for two-level pipelined implementation.



```

send_right  $\lambda r$ 
 $S \leftarrow \lambda^2 r^2$ 
do  $i = 1, k$ 
    fetch  $x_i$ ;  $S \leftarrow S + x_i^2$ ; send_right  $x_i$ ;
end do
 $r' \leftarrow \lambda r$ ;  $s \leftarrow \sqrt{S}$ ; send_right  $s$ ;
 $r \leftarrow s$ ;  $\sigma \leftarrow r' + s$ ; send_right  $\sigma$ .

```



```

 $S = 0$ ;
do  $i = 1, K + 1$ 
    fetch  $u_i$ ;
    if  $i = 1$  then
         $S \leftarrow S + u_i \lambda r$ ; send_right  $u_i$ ;
    else
        fetch  $x_{i-1}$ ;  $S = S + u_i x_{i-1}$ ; send_right  $u_i$ ;
    end if
    fetch  $s$ ;  $r' \leftarrow \lambda r$ ;  $t \leftarrow -S/s$ ;
    fetch  $\sigma$ ;  $r \leftarrow t$ ;  $d \leftarrow (r' - t)/\sigma$ ;
end do
do  $i = 1, k$ 
     $y_i = x_i - d u_{i+1}$ ; send_down  $y_i$ ;
end do

```

Figure 2.9: Operations of the processing cells

the same time, to obtain t ; then t is sent down and σ again arrives at the same time to compute d . To compute y_i of (2.8) in Step 3, we need some registers to store u_i and x_i temporarily. Since data from the next block is continuously being sent into the system, each internal cell needs $2(k + 3)$ registers to store u_i and x_i as indicated in Fig. 2.8. When d is available, y_i is then obtained one by one and sent down for further processing. For data from the next block, it goes through the same processing. When a new d is available in the internal cell, the corresponding x_i and u_i are already waiting in the registers. Therefore the vector operations are successfully pipelined down to the word level. This means that by using the modified HT algorithm, we have not only pipelined the SBHT arrays at the vector level but also at the word level. The input data is now skewed in the word level as shown in Fig. 2.8 rather than in the vector level as shown in Fig. 2.6.

Since the most time consuming operation of this two-level pipelined implementation is the square root operation which is also the critical operation in McWhirter's Givens rotation implementation, the throughput of this two-level pipelined implementation is as fast as that of the McWhirter's Givens array. However, with a longer pipeline, a longer system latency for the SBHT method is required. This is due to the fact that the registers of the internal cells have to be all filled before we can obtain the residual vector. For the SBHT RLS systolic array of order p , we have $(p^2 + 3p)/2$ internal cells, including the BPA. Thus, there are a total of $(p^2 + 3p)(k + 3)$ registers for the whole system. The system latency is given by $t_s = 2p(k + 4)$, which is linearly proportional to p and k . However, for the Givens rotation method, the system latency is only $t_s = 2p + 1$. Comparisons of both RLS arrays based on the SBHT and Givens rotation are summarized in Table 2.1. In general, the throughput

	Givens rotation	SBHT
Number of cells	$(p^2+3p)/2$	$(p^2+5p)/2$
Number of delay elements	p	$p(p-1)$
Number of registers	0	$(p^2+3p)(k+3)$
System latency	$2p+1$	$2p(k+4)$
Cell complexity	less	higher
Numerical stability	good	better

Table 2.1: Comparisons of the SBHT and Givens rotation methods

of the SBHT RLS systolic array is as fast as Givens rotation method. Of course, while the cell complexity of the SBHT array is higher, it does offer better numerical property [107]. A detailed backward error analysis carried out by Wilkinson showed that for an $n \times n$ matrix A , after $n(n-1)/2$ Givens rotations, the roundoff error in the upper-triangular matrix is in the order of $\mathcal{O}(\kappa_g n^{3/2} \mu \|A\|)$ [107, pp.138], while a series of $n-1$ HT's gives about $\mathcal{O}(\kappa_h n \mu \|A\|)$ [107, pp.160], with κ_g and κ_h being constants and μ a floating machine computation precision constant.

2.5 Constrained RLS Problems

In the above sections, we have dealt with an unconstrained RLS problem. The RLS systolic array considered there was motivated originally by the sidelobe canceller beamformation problem [78]. Other practical motivation could have come from the

adaptive filtering problem [32]. However, there are other signal processing applications which are modeled by a constrained RLS problem. The MVDR beamforming constitutes such an example [80, 85, 97]. It is interesting to determine whether a systolic array for an unconstrained RLS problem can also be used for a constrained RLS problem. In [80], McWhirter and Shepherd showed an extension of the unconstrained RLS array to the MVDR beamforming problem. Based on their approach, we shall also demonstrate the implementation of a MVDR beamformation problem using a SBHT RLS array.

The MVDR beamforming problem is to minimize

$$\xi^{(\ell)}(n) = \|\mathbf{X}(n)\mathbf{w}^{(\ell)}(n)\|_{\Lambda}, \quad \ell = 1, \dots, L, \quad (2.45)$$

subject to the linear constraints of

$$\mathbf{c}^{(\ell)T} \mathbf{w}^{(\ell)}(n) = \beta^{(\ell)}, \quad \ell = 1, \dots, L, \quad (2.46)$$

where L is the number of constraints. We are interested in the *a posteriori* residual vector

$$\hat{\mathbf{e}}_n^{(\ell)}(n) = \mathbf{X}_n^T \hat{\mathbf{w}}^{(\ell)}(n). \quad (2.47)$$

The optimal solution of the weight vector is known [80] to be given by

$$\hat{\mathbf{w}}^{(\ell)}(n) = \frac{\beta^{(\ell)} \mathbf{M}^{-1}(n) \mathbf{c}^{(\ell)}}{\mathbf{c}^{(\ell)T} \mathbf{M}^{-1}(n) \mathbf{c}^{(\ell)}} = \frac{\beta^{(\ell)} \mathbf{R}^{-1}(n) \mathbf{a}^{(\ell)}(n)}{\|\mathbf{a}^{(\ell)}(n)\|^2}, \quad (2.48)$$

where $\mathbf{M} = \mathbf{X}^T(n) \Lambda_k^2(n) \mathbf{X}(n)$ is the weighted covariance matrix, $\mathbf{R}(n)$ is the upper triangular matrix resulted from the QRD of the weighted data matrix $\Lambda_k \mathbf{X}(n)$, and

$$\mathbf{a}^{(\ell)}(n) = \mathbf{R}^{-T}(n) \mathbf{c}^{(\ell)}. \quad (2.49)$$

Therefore the optimal residual vector at time n is

$$\hat{\mathbf{e}}_n^{(\ell)}(n) = \frac{\beta^{(\ell)}}{\|\mathbf{a}^{(\ell)}(n)\|^2} \cdot \mathbf{X}_n^T \mathbf{R}^{-1}(n) \mathbf{a}^{(\ell)}(n). \quad (2.50)$$

A crucial step needed is for the efficient recursive updating of $\mathbf{a}^{(\ell)}(n)$. A novel approach was proposed for performing this updating [80]. Specifically, from (2.13), (2.14), and (2.49),

$$\begin{aligned}\mathbf{c}^{(\ell)} &= \mathbf{R}^T(n-1)\mathbf{a}^{(\ell)}(n-1) \\ &= \lambda^{-2} \begin{bmatrix} \lambda \mathbf{R}^T(n-1) & \vdots & \mathbf{0}^T & \vdots & \mathbf{X}_n \end{bmatrix} \begin{bmatrix} \lambda \mathbf{a}^{(\ell)}(n-1) \\ \text{---} \\ \lambda \mathbf{b}^{(\ell)}(n-1) \\ \text{---} \\ \mathbf{0} \end{bmatrix},\end{aligned}\quad (2.51)$$

where $\mathbf{b}^{(\ell)}(n-1)$ is an arbitrary $((n-1)k-p) \times 1$ vector. Then from *Lemma 2.1*, (2.13), and (2.14), we have

$$\begin{aligned}\mathbf{c}^{(\ell)} &= \lambda^{-2} \begin{bmatrix} \lambda \mathbf{R}^T(n-1) & \vdots & \mathbf{0}^T & \vdots & \mathbf{X}_n \end{bmatrix} \mathbf{H}^T(n) \mathbf{H}(n) \begin{bmatrix} \lambda \mathbf{a}^{(\ell)}(n-1) \\ \text{---} \\ \lambda \mathbf{b}^{(\ell)}(n-1) \\ \text{---} \\ \mathbf{0} \end{bmatrix} \\ &= \mathbf{R}^T(n) \cdot \lambda^{-2} (\lambda \mathbf{H}_{11}(n) \mathbf{a}^{(\ell)}(n-1)).\end{aligned}\quad (2.52)$$

Thus, $\mathbf{a}^{(\ell)}(n) = \lambda^{-2} (\lambda \mathbf{H}_{11}(n) \mathbf{a}^{(\ell)}(n-1))$ can be obtained by updating $\mathbf{a}^{(\ell)}(n-1)$ in a way similar to that $\mathbf{u}(n)$ is obtained by updating $\mathbf{u}(n-1)$ using (2.37). The only differences are the input for updating $\mathbf{a}^{(\ell)}(n-1)$ is a zero vector and a scaling factor λ^{-2} . Due to the structure of \mathbf{H} in *Lemma 1*, the vector $\mathbf{b}^{(\ell)}(\cdot)$ plays no role in the updating of $\mathbf{a}^{(\ell)}(\cdot)$. Furthermore, from (2.32) and (2.34), we have

$$\hat{\mathbf{e}}_n(n) = \mathbf{X}_n^T \mathbf{R}^{-1}(n) \mathbf{u}(n) - \mathbf{y}_n. \quad (2.53)$$

From (2.37), we see that $\mathbf{u}(n)$ results from the update of $[\mathbf{y}(n-1) \vdots \mathbf{y}_n]$, where \mathbf{y}_n is the new input. Now replacing $\mathbf{u}(n)$ with $\mathbf{a}^{(\ell)}(n)$ and \mathbf{y}_n with a zero vector, we have

$$\hat{\mathbf{e}}_n(n) = \mathbf{X}_n^T \mathbf{R}^{-1}(n) \mathbf{a}^{(\ell)}(n), \quad (2.54)$$

and from (2.50), we then obtain

$$\hat{\mathbf{e}}_n^{(\ell)}(n) = \frac{\beta^{(\ell)}}{\|\mathbf{a}^{(\ell)}(n)\|^2} \cdot \hat{\mathbf{e}}_n(n). \quad (2.55)$$

This equation reveals that by the proper scaling of $\hat{\mathbf{e}}_n(n)$, which can be obtained from the SBHT RLS systolic array, we can obtain the *a posteriori* residual vector, $\hat{\mathbf{e}}_n^{(\ell)}(n)$, of the MVDR beamformation. Fig. 2.10 shows an extension of the SBHT RLS array for the new problem. Now one more data channel is needed for the RA to pipeline cumulation of $\|\mathbf{a}^{(\ell)}(n)\|^2$, and the scaling of the residual vector is done at the bottom of the RA when a new $\|\mathbf{a}^{(\ell)}(\cdot)\|^2$ is available. Each RA/BPA pair in Fig. 2.10 represents one of the K constraints. The optimal *a posteriori* residual vector of each linear constraint is obtained at the output of the corresponding backward propagation array.

As pointed out in [80], there are two ways to initialize the array. One method is to set $\mathbf{R}(0) = \delta \mathbf{I}$, where δ is a small scalar, and thus from (2.49), $\mathbf{a}^{(\ell)}(0) = \delta^{-1} \mathbf{c}^{(\ell)}$, $\ell = 1, \dots, L$. Another method is to obtain $\mathbf{R}(n)$ to some time n , then use (2.49) to obtain $\mathbf{a}^{(\ell)}(n)$. The details of a two-mode operation required for this initialization procedure are also considered in [80].

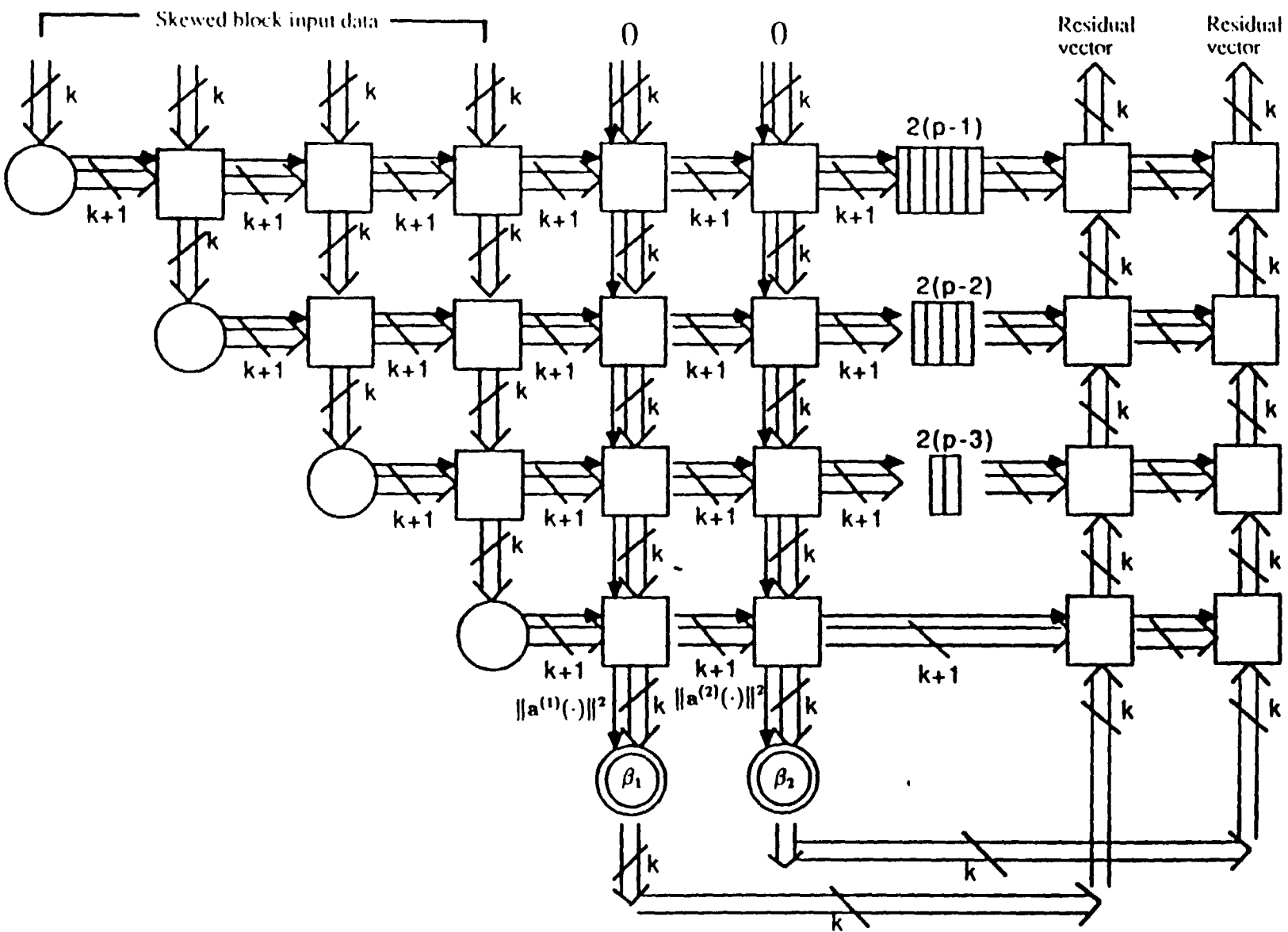


Figure 2.10: The MVDR beamforming systolic array

Chapter 3

Real-Time Algorithm-Based Fault-Tolerance for QRD RLS Systolic Array

In this chapter, we propose a new algorithm-based fault-tolerant method derived from the inherent nature of the QR least-squares systolic algorithm. Since the residuals of different desired responses can be computed simultaneously, an artificial desired response can be designed to detect an error produced by a faulty processor. We show that if the artificial desired response is designed as some proper combinations of the input data, the output residual of the system will be zero if there is no fault. However, any occurring fault in the system will cause the residual to be non-zero and the fault can be detected in real-time. Once the fault has been detected, the system enters into the fault diagnosis phase from the concurrent error detection phase. Two methods, the flushing fault location and the checksum encoding methods, can be used to diagnose the location of the faulty row. When the faulty row is determined,

this row and the associated column with the same boundary cell are eliminated by a reconfiguration operation. Then the system degrades in a graceful manner which is generally acceptable for many least-squares applications. Those eliminated processors enter into a self-checking phase, and when the transient fault condition is removed, a reconfiguration is performed to resume the normal full order operation. The analysis of error propagation and recovery latency is also considered in this chapter.

This chapter is organized in the following manner. In section 3.1, we review some previous works in algorithm-based fault-tolerance and their advantages and disadvantages. In section 3.2, we address the fault model. Various fault-tolerance schemes, including concurrent error detection, fault diagnosis, and order degraded reconfiguration are presented in section 3.3. In section 3.4, we prove the residual method is robust and discuss some important latencies. Finally, in section 3.5, we present some conservation tests which may also be used to detect faults.

3.1 Algorithm-Based Fault-Tolerance

For a given algorithm, some inherent nature of that algorithm can be used to develop a highly efficient specific fault-tolerant technique denoted as *algorithm-based fault-tolerance*. The term *algorithm-based* means it is an algorithm-specific and not a general scheme that can be applied to all general problems. An inherent property of many matrix computations is that checksums preserved after the computations. This observation has led to the discovery of algorithm-based fault-tolerance which was first studied at University of Illinois at Urbana. A recently reported algorithm-based fault-tolerant technique, called checksum (and weighted checksum) encoding scheme proposed by Hwang, Abraham, Jou, Chen et al., has evolved from the study

of VLSI matrix computations [3,4,5,6]. This scheme belongs to the category of information redundancy [21]. Since few hardware and time redundancies are necessary, it is promising for its low-cost and low overhead for VLSI/WSI multiprocessor systems. Many applications of the checksum (or weighted checksum) scheme have been successfully applied to various signal processing and linear algebra operations [13]. The major drawback of the checksum scheme proposed in [6] is that the system throughput will be slowed down because the system clock has to be extended long enough to accommodate the longer signal path of the non-local interconnection caused by the checksum scheme. Unfortunately, local connection is one of the basic desirable requirements of implementations. Define the checksum vector

$$\underline{e}^T = [1, 1, \dots, 1]$$

and weighted checksum vector

$$\underline{e}_w^T = [w_1, w_2, \dots, w_n],$$

the column, row and full weighted checksum matrices A_{cw} , A_{rw} , and A_{fw} of a square n -by- n matrix A as

$$A_{cw} = \begin{bmatrix} A \\ \underline{e}^T A \\ \underline{e}_w^T A \end{bmatrix}, \quad (3.56)$$

$$A_{rw} = \begin{bmatrix} A & A\underline{e} & A\underline{e}_w \end{bmatrix}, \quad (3.57)$$

$$A_{fw} = \begin{bmatrix} A & A\underline{e} & A\underline{e}_w \\ \underline{e}^T A & \underline{e}^T A\underline{e} & \underline{e}^T A\underline{e}_w \\ \underline{e}_w^T A & \underline{e}_w^T A\underline{e} & \underline{e}_w^T A\underline{e}_w \end{bmatrix}. \quad (3.58)$$

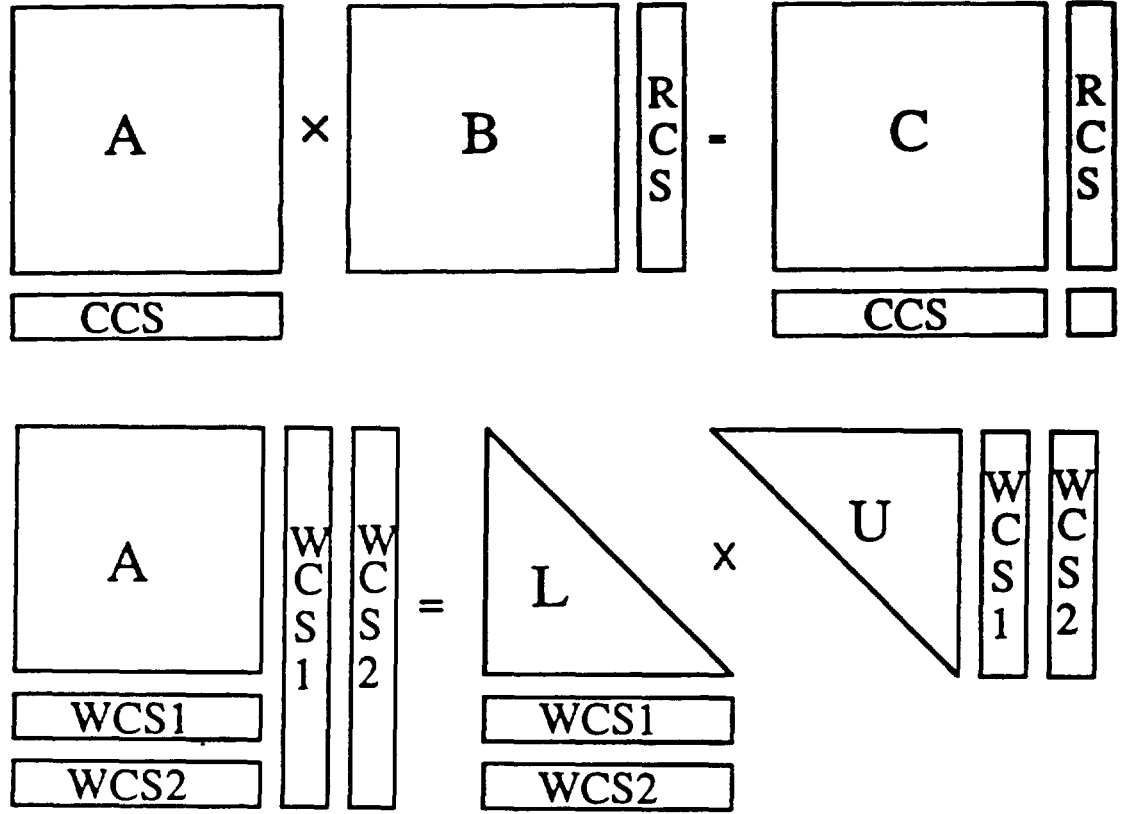


Figure 3.11: Matrix-matrix multiplication and LU decomposition with (weighted) checksum encoding

If there is any fault occurring during the computation, the checksum criterion is not met and thus the fault is detected. The checksum and weighted checksum fault-tolerance schemes have been proposed by Huang, Jou, Chen, and Abraham [14, 15, 33, 41] to various signal processing and linear algebra operations such as matrix multiplication and inversion, QR decomposition, LU decomposition and singular value decomposition (SVD) *et al.*. For examples, Fig. 3.11 shows the applications of checksum scheme to matrix-matrix multiplication and LU decomposition. The weighted checksum scheme can be further used to correct errors [41, 2]. The number of error to be corrected depends on how many weighted checksum vectors are

used. The major advantages of (weighted) checksum scheme are low-cost and low overhead for error detection and correction of multiple processor systems. However, the disadvantages are:

1. There is the problem to distinguish roundoff errors from those caused by failures [47];
2. If the intermediate results of each processor are continuously propagated to other processors such as QRD LS systolic array, the system throughput will be slowed down because the system clock has to be extended long enough to accomodate the longer signal path of the non-local interconnections caused by the checksum scheme.

There are some difficulties in applying checksum schemes to system problems such as communication data equalization, and radar/sonar adaptive antenna array, where the signal arrives continuously and high throughput rate is required. As an example, for a recursive QRD LS systolic array, the data is coming in row by row. As pointed out in [17], the QRD of a row checksum encoding matrix A_r results in a row checksum encoding upper triangular matrix R_r . That is, $A_r = QR_r$. If the checksum scheme is used to detect error, during the recursive operation, when a new upper triangular matrix R is formed, to obtain a new checksum for each row, we need to sum up each elements of the corresponding row. Suppose only local connection, which is one of the basic desirable requirements of VLSI implementations, is allowed, to prevent severe throughput degradation, a new channel used to pipe out the partial sum of matrix R has to be built. While a new content of each cell is available, this content has to be added to the partial checksum sent from the previous cells through the new checksum

channel and then the partial checksum has to be passed to the right for the next cell. With these requirements and operations, not only the complexity is increased but also the throughput is hindered. Even when global communications of the processors are allowed, the checksum scheme for fault detection still reduces the throughput of the system as pointed out above.

Most recently, [4] has proposed a new algorithm-based fault-tolerant technique applicable to the recursive LS triangular systolic array. With the addition of one extra column of processor array, errors in the tri-array can be detected by observing the scalar output of this column array. While there are some similarities between the results reported in [4] and that considered in this chapter (as well as that in an earlier version of our paper [65]), there are much differences in assumptions, techniques, and results between these two approaches. In any case, these two works were performed independently of each other.

3.2 Fault Model

As the VLSI technology progresses, the geometric features become smaller. Any defect affecting a given part of the circuitry may cause an entire module or a logic block to become faulty and to produce arbitrary errors. Thus, the traditional gate-level single stuck-at fault model is no longer appropriate for VLSI/WSI system. A cell or module is allowed to produce arbitrary errors if any part of the cell is under failures [17]. However, we assume that at most one cell can be faulty at a given short period of time. This is based on the assumption that the system reliability is such that the mean time between failures is long enough and the probability of more than one fault occurring is very small. Some basic assumptions we need are as follows:

1. If any part of the cell become faulty, the whole cell will not function correctly;
2. The probability of the communication links and registers failing is very small and thus negligible [55].

The second assumption is reasonable since these components are typically much simpler and smaller than the processing cells themselves [55]. In addition, they can be implemented conservatively with high redundancy or with self-testing circuitry to mask a possible fault.

3.3 Real-Time Fault-Tolerance

While a demanding system is usually expected to operate under a high performance condition, when a fault has occurred, a slightly degraded performance is often acceptable under the circumstance. For a recursive QRD LS systolic array operating under a normal fault-free condition, the optimum LS solution is attained. A reduced order degraded LS solution can still yield a reasonable performance if the row and column with the faulty processor can be eliminated from the computation. This concept leads to our proposed design of a gracefully degraded fault-tolerance approach for the QRD LS systolic array.

3.3.1 Concurrent Error Detection - Residual Method

An inherent nature of the QRD LS systolic algorithm is that for a given data matrix A , the minimization of $\|Aw_i(n) - \underline{y}_i(n)\|_\Lambda$ for many desired response vectors $\underline{y}_i, i \in I$, can be performed concurrently by appending some more RA's to the systolic array such as shown in Fig. 3.12. The output of the i^{th} RA, $e_i(n)$, is the minimal residual

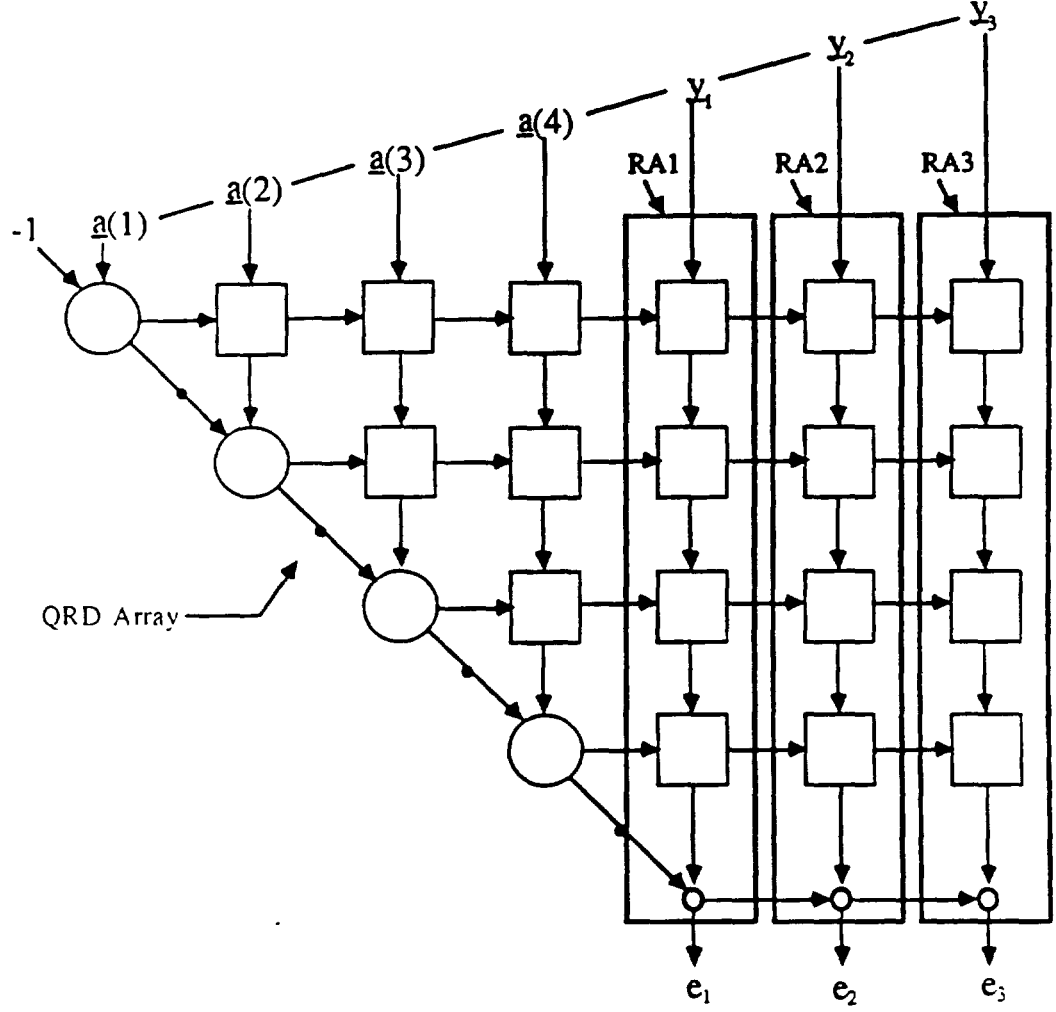


Figure 3.12: Recursive LS systolic array with multiple desired responses

of $\underline{u}^T(n)\underline{\hat{w}}_i(n) - d_i(n)$, where $d_i(n)$ is the n^{th} desired response of i^{th} input vector \underline{y}_i . Let \underline{y}_0 belong to the column space of A . That is, $\underline{y}_0 \in \text{span}\{\underline{a}(i), 1 \leq i \leq p\}$, then $\underline{y}_0 = \sum_{j=1}^p c_j \underline{a}(j)$. The optimal LS residual and the associated weight vector for \underline{y}_0 are $e_0(n) = 0$ and $\underline{w}_i(n) = [c_1, c_2, \dots, c_p]^T$, for $n \geq p$. The actual selection of $\{c_1, c_2, \dots, c_p\}$ will be given later. Various extensions of these fundamental properties of the optimum residual of a LS estimation problem form the basis of the proposed *residual method* approach toward concurrent error detection. Denote \underline{y}_0 to be the **artificial desired response (ADR)** and the associated RA as the **error detection**

array (EDA).

Lemma 3.1 *Given the ADR $\underline{y}_0 = \underline{a}(p)$, the contents of the EDA are identical to the contents of the p^{th} column array of the QRD triangular array, and the optimal residual e_0 , the output of the p^{th} cell of the EDA, is always zero.*

Proof: Both arrays are $p \times 1$ vector. The first $p - 1$ elements of both arrays are identical given by the fact that the same data are rotated by the same c_i and s_i generated by boundary cells $PE_{ii}, 1 \leq i \leq p - 1$, where PE_{ii} denoted the processor at position (i, i) . Thus the outputs of the $(p - 1)^{th}$ cells of both arrays are identical. Initially, the contents of the p^{th} cells of both array are zeros. Let the first non-zero output of $(p - 1)^{th}$ cells be x , by the update equations of both cells, the first non-zero contents of both p^{th} cells equal x . If the second non-zero output of $(p - 1)^{th}$ cells is z , the updated content of the p^{th} boundary cell equals $\sqrt{\lambda^2 x^2 + z^2}$ and that of the p^{th} cell of the EDA is $s_p z + c_p \lambda = \sqrt{\lambda^2 x^2 + z^2}$, where $s_p = z / \sqrt{\lambda^2 x^2 + z^2}$ and $c_p = \lambda x / \sqrt{\lambda^2 x^2 + z^2}$. Therefore, the contents of both array are identical. Since the rotation coefficients, c_p and s_p , generated by PE_{pp} boundary cell are proportional to x and z respectively, the output of the p^{th} cell of the EDA, e_0 , is $c_p z - s_p x$ which is always zero. \square

If there is a fault in either the p^{th} column of the array or the EDA, these contents are no longer identical and then lead to a non-zero e_0 . Thus a fault is detected. However, if there is any fault outside of these two arrays, then the errors produced by that fault will affect both of these arrays in the same manner (i.e., contents of both arrays are still identical) and resulting in a zero e_0 . Thus, these faults will not be detected by the $\underline{y}_0 = \underline{a}(p)$ design. Clearly, we can generalize the above results by the following Lemma.

Lemma 3.2 *Given the ADR $\underline{y}_0 = \underline{a}(k)$, for $1 \leq k \leq p$, the contents of the k^{th} column array of the QRD triarray and the first k cells of the EDA are identical. The output of the k^{th} cell of the EDA is zero. The contents of cell l , $k + 1 \leq l \leq p$, of the error detection array are all zeros.*

Corollary 3.1 *A fault occurring outside of the k^{th} column of the QRD triarray and the EDA will not be detected if the ADR is designed as $\underline{y}_0 = \underline{a}(k)$.*

Proof: From the previous discussion, a fault occurring in the i^{th} , $1 \leq i \leq k - 1$, column of QR array will not be detected. From *Lemma 3.2*, the output of the k^{th} cell and the contents of cell l , $k + 1 \leq l \leq p$, of the EDA are all zeros. Thus, any fault occurring to the i^{th} , $k + 1 \leq i \leq p$, column of QR array will be masked by these zeros. The optimal residual e_0 is always zero unless there is any inconsistency between the k^{th} column of the QR array and the EDA. \square

From all the above observations, by selecting \underline{y}_0 properly as given in the following theorem, we can detect the presence of a fault in any location of the system.

Theorem 3.1 (Concurrent Error Detection Theorem) *Consider the selection of the artificial desired response $\underline{y}_0 = \sum_{i=1}^p \underline{a}(i)$. If there is no fault in the system, then the output of the EDA with \underline{y}_0 as an input yields $e_0 = 0$. If there is a fault in the system, then $e_0 \neq 0$.*

Proof: From *Lemma 3.2*, each $\underline{a}(i)$ is "zeroed out" by the i^{th} cell of the EDA. Any error produced by a faulty processor, say in the j^{th} column of the QR array, will not be zeroed out by the j^{th} cells of the EDA. The output of the j^{th} cell is then non-zero and propagates down to the output. Therefore, whenever $e_1 \neq 0$, there is a fault in the system. \square

The ADR $\underline{y}_0 = \sum_{i=1}^p \underline{a}(i)$ is obtained by implementing a top row encoding array (EA) consisting of p summing cells as shown in Fig. 3.13. The response array (RA), with the desired response \underline{y} as input and e as output, located at the right of the EDA is incorporated with the system to produce the desired residual. Once $e_0 \neq 0$, which indicates the system had a fault, then $e(n)$ is considered to be in error and will not be used. The error detection is thus achieved in real-time.

Example 3.1: An adaptive filter using QRD LS systolic array with order $p = 3$ is simulated. In between $t = 25$ and $t = 35$, a fault occurred in cell PE_{23} in such a way that random noise within range $[-1, 1]$ is generated. From Table 3.2, we can see, due to the propagation delay which will be considered in Section 3.4.2, from $t = 28$ to $t = 38$, $e_0 \neq 0$ results from errors generated by the faulty cell. The optimal residuals in e from $t = 28$ to $t = 38$ are then considered faulty. After $t = 39$, e_0 then decays down due to the adaptive nature of the algorithm. Fig. 3.15 plots $|e_0|$ versus t and shows the adaptive effect of the algorithm. A threshold device can be used with e_0 to provide a decision on the size of the error that can be tolerated. Fig. 3.16 shows the $|e_0|$ in Fig. 3.15 with threshold set at 0.3. A generalization of the proposed scheme is stated below.

Theorem 3.2 (Generalized Error Detection Theorem) *Any fault occurring in the system can be detected if the ADR is given by $\underline{y}_0 = \sum_{i=1}^p c_i \underline{a}(i)$, where $c_i \neq 0, 1 \leq i \leq p$. \square*

The simplest ADR that can detect fault is indeed a checksum encoded data (given by Theorem 3.1) which is a special case of the set of ADR given by Theorem 3.2. However, unlike the checksum fault detection scheme in [17, 47], Theorem 3.1 and

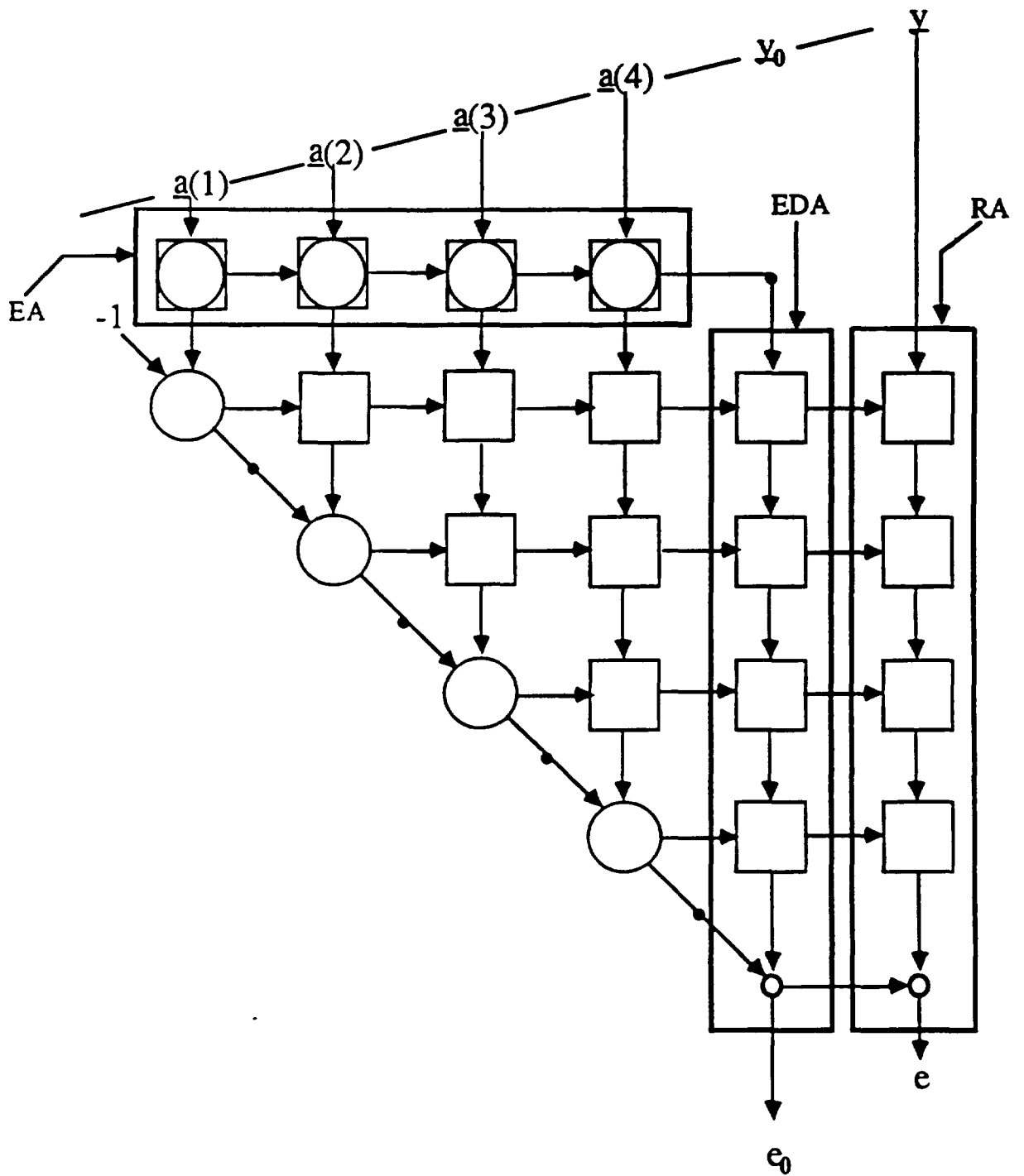
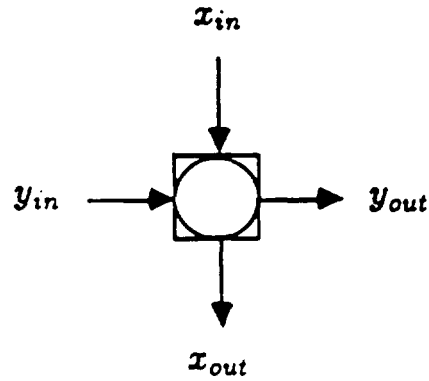


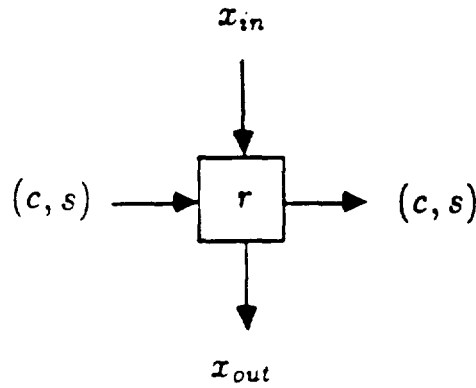
Figure 3.13: Fault-tolerant recursive LS systolic array based on linear combination of input data in top row array feeding into column error detection array with output fault indication variable e_0 .

(1) Encoding Cell



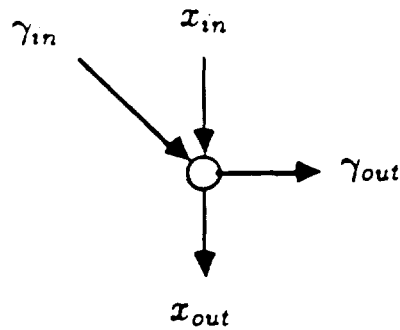
$$\begin{aligned} x_{out} &\leftarrow x_{in} \\ y_{out} &\leftarrow x_{in} + y_{in} \end{aligned}$$

(2) Internal Cell



If $(c = 1 \text{ and } s = 0)$ then
 $x_{out} \leftarrow x_{in}, \quad r \leftarrow \lambda r$
 else
 $x_{out} \leftarrow cx_{in} - s\lambda r$
 $r \leftarrow sx_{in} + c\lambda r$
 end if

(3) Final Cell



$$\begin{aligned} x_{out} &\leftarrow \gamma_{in} x_{in} \\ \gamma_{out} &\leftarrow \gamma_{in} \end{aligned}$$

Figure 3.14: Processing cells of fault-tolerant systolic array.

t=1	e0=0.000000	e=-0.000000
t=2	e0=0.000000	e=-0.000000
t=3	e0=0.000000	e=-0.000000
t=4	e0=0.000000	e=-0.000000
t=5	e0=0.000000	e=-0.000000
t=6	e0=0.000000	e=-0.000000
t=7	e0=-0.000000	e=-0.000000
t=8	e0=-0.000000	e=-0.000000
t=9	e0=-0.000000	e=-0.000000
t=10	e0=-0.000000	e=-0.000000
t=11	e0=0.000000	e=-0.000000
t=12	e0=0.000000	e=-0.000684
t=13	e0=0.000000	e=-0.153411
t=14	e0=0.000000	e=-0.138241
t=15	e0=0.000000	e=-0.155885
t=16	e0=-0.000000	e=0.065978
t=17	e0=-0.000000	e=0.507390
t=18	e0=-0.000000	e=-0.000090
t=19	e0=0.000000	e=-0.704896
t=20	e0=-0.000000	e=0.018910
t=21	e0=0.000000	e=-0.053229
t=22	e0=0.000000	e=-0.787624
t=23	e0=-0.000000	e=0.015445
t=24	e0=0.000000	e=-0.287406
t=25	e0=0.000000	e=-0.677166
t=26	e0=0.000000	e=-0.050505
t=27	e0=0.000000	e=0.817616
<hr/>		
t=28	e0=-0.966001	e=-0.553705
t=29	e0=-0.611989	e=0.325937
t=30	e0=1.083665	e=0.102332
t=31	e0=-0.506277	e=-0.100179
t=32	e0=-0.391151	e=-0.430207
t=33	e0=0.538918	e=0.689049
t=34	e0=-0.426679	e=-0.220855
t=35	e0=-0.558384	e=0.312747
t=36	e0=0.975315	e=-0.117668
t=37	e0=-0.748951	e=0.038787
t=38	e0=0.144688	e=0.219978
<hr/>		
t=39	e0=-0.053715	e=0.386494
t=40	e0=-0.014142	e=0.095640

Table 3.2: The output of e_0 and e of an adaptive QRD LS filter

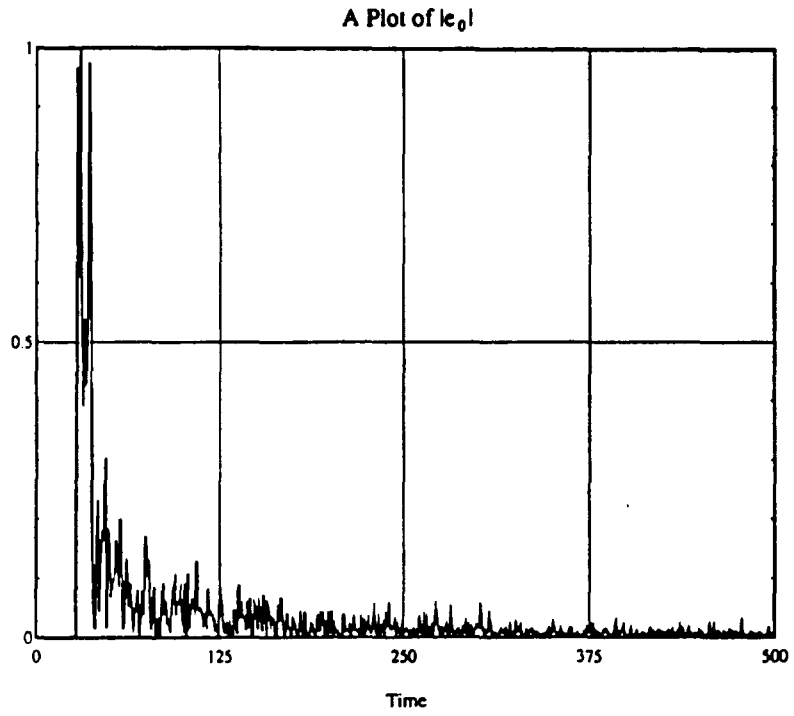


Figure 3.15: Plot of $|e_0|$ of an adaptive QRD LS filter with order $p = 3$ when a fault occurred in PE_{23} from $t = 25$ to $t = 35$.

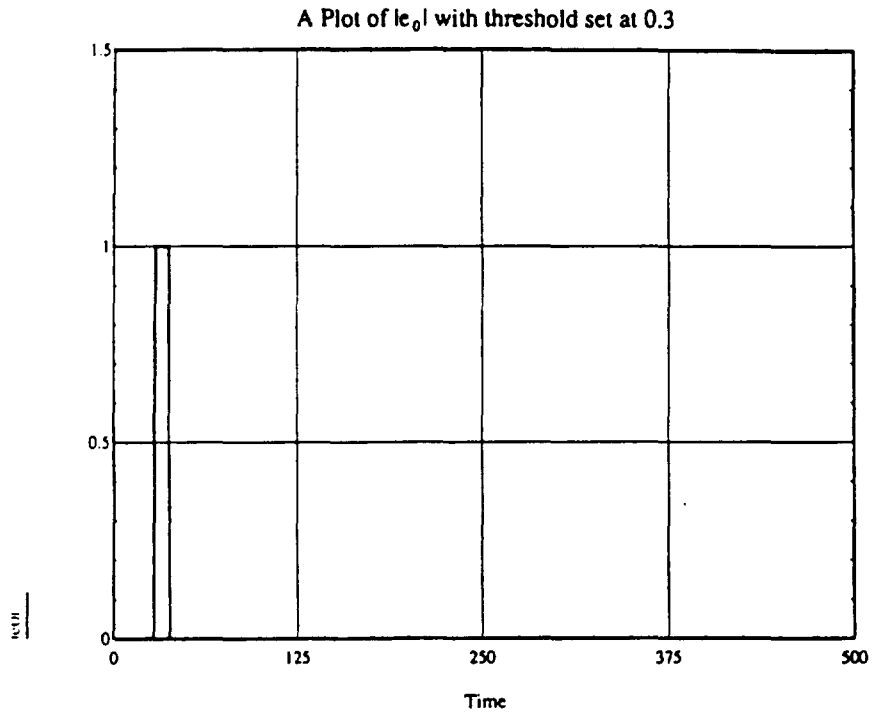


Figure 3.16: Plot of $|e_0|$ in Fig.3(a) with threshold set at 0.3.

3.2 provide a real-time fault detection scheme using the inherent nature of QRD RLS systolic array.

Optimal Efficiency

Since a column of linear EDA and a row of linear EA are required, the complexity of this fault detection scheme is $2p$. That is, $2p$ redundant processors is required. Compare to the complexity of the triangular QR LS array, $(p^2 + 3p)/2$, it is a cost-effective real-time fault detection scheme. Here we do not count the final output multiplier cells in the EDA and RA.

We define the *hardware efficiency* Ω_h to be the ratio of the hardware cost of implementing the algorithm to the cost of implementing the algorithm with an error-detection capability. We see

$$\Omega_h(p) = \frac{p^2 + 3p}{p^2 + 7p}. \quad (3.59)$$

Thus $1/2 \leq \Omega_h(p) \leq 1$ since a single error can be detected by duplicating the hardware. When $\Omega_h(p) = 1$, we say the error-detection scheme is *most hardware efficient*. Define the *time efficiency* Ω_t to be the ratio of the time to implement the algorithm and the time to implement the algorithm incorporating the error-detection scheme. Obviously, time efficiency is bounded by $0 \leq \Omega_t \leq 1$. When $\Omega_t = 1$, we say the error-detection scheme is *most time efficient*. If an error-detection scheme is both most hardware and time efficiency, then it is said to be *optimal*. For the proposed residual method, clearly $\lim_{p \rightarrow \infty} \Omega_h(p) = 1$ as shown in Fig. 3.17. That is, it is asymptotically most hardware efficient. However, the time efficiency $\Omega_t(p) = 1, \forall p$, so that it is also most time efficient. Therefore, the residual method is an asymptotically optimal error-detection scheme for the recursive LS systolic array.

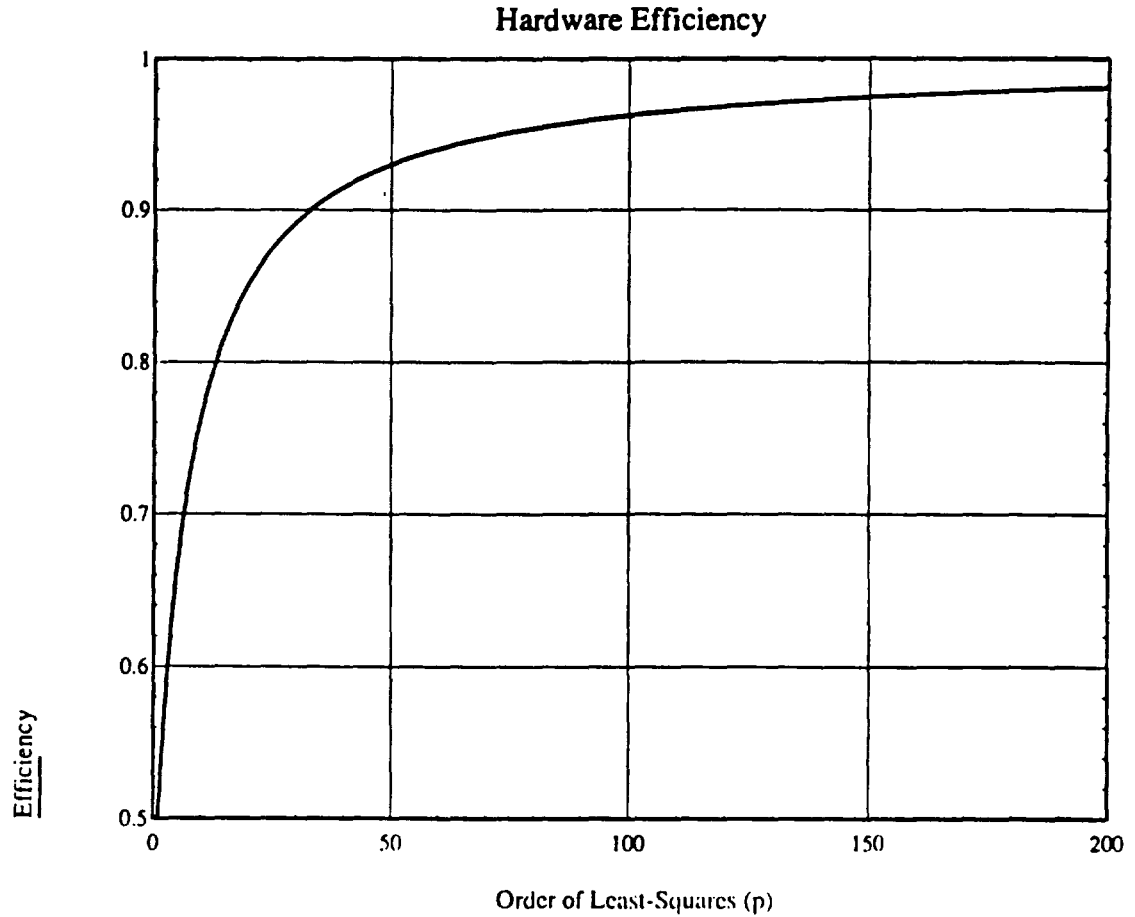


Figure 3.17: Plot of the hardware efficiency of the residual method versus the order of the LS estimation.

3.3.2 Fault Diagnosis

When a fault is detected, the system leaves the concurrent error detection phase and enters the fault diagnosis phase. The main purpose of this phase is to find the faulty processor row. Either of two methods, the flushing fault location (FFL) method or the checksum encoding (CSE) method, can be used to diagnose and locate the faulty row. The FFL method is developed under the assumption that only the residual output e_0 can be accessed externally, while the CSE method assumes that all the cells of the EDA can be accessed.

Flushing Fault Location Method

During the concurrent error detection phase, a fault is detected based on unknown values of the incoming data in $\underline{a}(k)$, $1 \leq k \leq p$, and contents of all the cells. However, in the FFL method, we will control the desired incoming data as well as the contents of the tri-array and the EDA, in order to obtain an appropriate value in e_0 to locate the faulty processor row. In the FFL method, we do not use the operations of the RA cells or the EA cells. From (2.3), the weight vector $\underline{\hat{w}} = (\hat{w}_1, \hat{w}_2, \dots, \hat{w}_p)$ can be solved by using the back substitution method of

$$\hat{w}_i = \frac{P_i - \sum_{j=i+1}^p r_{ij} \hat{w}_j}{r_{ii}}, \quad i = p, p-1, \dots, 1, \quad (3.60)$$

where P_i and r_{ij} are elements of vector \underline{P} and matrix R . A linear array to performed the back substitution as in [29] can be prevented by using the *weight flushing technique* [106].

In a fault-free triangular LS systolic array, by "freezing" the QRD upper triangular matrix $R(n)$ and the associated column vector $\underline{P}(n)$ in (2.3), the optimum solution $\underline{\hat{w}}(n)$ can be "flushed" out sequentially with a skewed identity matrix input. In these operations, the internal cells (defined in Fig. 3.17) in a given row, act as pure bypass elements with no Givens rotations, when the input to the boundary cell of that row is zero and $c = 1$ and $s = 0$ are being propagated. For example, let $p = 3$ and $\underline{P} = (P_1, P_2, P_3)$, then

$$\begin{aligned} \hat{w}_3 &= P_3/r_{33}, \\ \hat{w}_2 &= (P_2 - r_{23}\hat{w}_3)/r_{22}, \\ \hat{w}_1 &= (P_1 - r_{12}\hat{w}_2 - r_{13}\hat{w}_3)/r_{11}. \end{aligned} \quad (3.61)$$

However, due to errors generated by the faulty processor, various parts of $R(n)$

and \underline{P} stored in the array are no longer correct. A new test triangular matrix T and a test vector \underline{P}_t are load to the tri-array and EDA respectively. The values of T and \underline{P}_t can be either pre-stored or distributed by an host computer when a fault is detected. Specifically, define $T = [\underline{t}_1, \underline{t}_2, \dots, \underline{t}_p]$ as a $p \times p$ all-1's upper triangular matrix, where \underline{t}_i is a $p \times 1$ vector, and \underline{P}_t as a $p \times 1$ all-1's vector. Since $\underline{P}_t = \underline{t}_p$, the optimal solution vector $\hat{\underline{w}}_0 = [0, 0, \dots, 1]$, as given by (2.3). One of the reasons for the selection of these all 1's in T and \underline{P}_t is to reduce memory requirement. Only one-bit register is required for each cell or distribution requirement.

With T and \underline{P}_t frozen, consider a skewed $p \times p$ identity matrix input to the first p columns and all zeros input to the EDA. In the absence of a fault, components of the optimum solution vector, $\hat{\underline{w}}_0 = [0, 0, \dots, 1]$, are outputted sequentially at e_0 [106]. Denote $\underline{\phi}_i^T = [0, \dots, 0, 1, 0, \dots, 0]$ as a $1 \times p$ vector of all zeros except for an one at the i^{th} position, representing the i^{th} row of the skewed identity matrix input to the array. Then the output $e_0(1)$ in response to $\underline{\phi}_1^T$ is processed by all the cells from the first to the p^{th} row of the array. In general, $e_0(i)$ is the response to $\underline{\phi}_i^T$ (i.e. $e_0(i) = \underline{\phi}_i^T \hat{\underline{w}}_0$), and is processed by all the cells from the i^{th} to the p^{th} row. As considered above, in the absence of a fault, $e_0(i) = 0$, $1 \leq i \leq p-1$, and $e_0(p) = 1$. However, with a fault in the k^{th} row, then $e_0(i) \neq 0$, $1 \leq i \leq k$, but responses due to $\underline{\phi}_j^T$ for $k+1 \leq j \leq p$, encountering only fault free processing cells from the j^{th} to the p^{th} row, will yield the correct value. This property can be used to locate the faulty row in the FFL method.

Theorem 3.3 (Flushing Fault Location Theorem) *When a fault is detected and the system enters the fault diagnosis phase, both the EA and RA arrays are made in-operative and all 1's are loaded into the contents of the processor cell. A skewed identity $p \times p$ matrix is flushed into the system with all zeros input to the EDA. The*

EDA output $e_0(i), 1 \leq i \leq p$ is obtained sequentially. Assume the first zero output at $e_0(k+1) = 0$, occurs for some $k, 1 \leq k \leq p-2$, then the faulty processor is in the k^{th} row. If there is no such k for $e_0(k+1) = 0, 1 \leq k \leq p-2$, and $e_0(p) = 1$, then the $(p-1)^{th}$ row is the faulty row; otherwise, with $e_0(p) \neq 1$, the p^{th} row is the faulty row. \square

Example 3.2: A QRD RLS array with order $p = 5$ is considered. Suppose a fault has occurred in PE_{34} . When a skewed 5×5 identity matrix is flushed into the system, due to the randomly generated noise from the faulty cell PE_{34} , the outputs from e_0 are given by $[0.2127, -0.5714, 0.7453, 0., 1.]$. In the absence of an error, the outputs should be $[0., 0., 0., 0., 1.]$. Since the first three elements are erroneous, based on *Theorem 3.3*, the faulty cell is in the third row.

It is obvious the flushing of ϕ_1 is unnecessary since computations involving the entire QR array are definitely incorrect in the fault diagnosis phase.

Checksum Encoding Method

The basic assumption of the CSE method is that all the cells of the EDA can be accessed. Further more, all the contents of the tri-array can be piped out in the diagnosis phase. In this chapter, instead of using the CSE as a fault detector as in [17], it is used to diagnose fault location when a fault has been detected. The disadvantages of the checksum scheme for real-time application is thus prevented. It has been shown in [17, 47] that the QRD of a row checksum matrix A_r results in a row checksum upper triangular matrix R_r . Let $r_{ij}(n)$ be the content of processor PE_{ij} of the tri-array at time n and $P_i(n)$ be the content of the i^{th} processor of the EDA at time n .

Theorem 3.4 (Checksum Encoding Theorem) *Given the artificial desired response $y_0 = \sum_{i=1}^p \alpha_i \underline{a}(i)$, for $\alpha_i \neq 0$, the checksum*

$$\sum_{k=0}^{p-i} \alpha_i r_{i(i+k)}(n+k) = P_i(n+p-i+1), \quad (3.62)$$

holds for $i = 1, 2, \dots, p$, $n = p, p+1, p+2, \dots$, if no fault has occurred. If there is an m such that the checksum does not hold for $m \leq i \leq p$, then there is a fault in the system and the faulty processor is in the first row that does not meet the checksum. \square

The time indices are introduced to describe the time difference for a given row input of data to the array. Each column of inputs, say $\underline{a}(k)$, is zeroed out by the k^{th} cell of EDA. Thus the content of the k^{th} cell of EDA is affected only by $\underline{a}(i)$, $k \leq i \leq p$. Therefore, if there is a faulty processor, say in row m , then all the rows below do not satisfy the checksum because of the error produced by the faulty one which cannot be zeroed out by the m^{th} cell of EDA.

Example 3.3: Consider a $[R : \underline{P}_0]$ matrix of intermediate results for a QRD RLS array with order $p = 4$,

$$\begin{bmatrix} 0.7930 & 0.7462 & 0.4655 & 0.9774 & : & 2.9821 \\ 0. & 1.2973 & 1.0816 & 1.0379 & : & 3.4168 \\ 0. & 0. & 0.2729 & 0.3643 & : & 1.7132 \\ 0. & 0. & 0. & 0.1675 & : & 0.8375 \end{bmatrix},$$

where the $(i, i+k)$ element, $r_{i(i+k)}$, of R , takes the value at time $n+k$ due to time skewing of the input. That is, $r_{i(i+k)}(n+k)$. The i^{th} element of \underline{P} takes the value at time $n+5-i$. That is, $P_i(n+5-i)$. As we can see, starting at the third row, the checksums are no longer consistent for each row. From *Theorem 3.4*, the faulty cell is in the third row.

Unlike the FFL method, the CSE method cannot stop the concurrent error detection phase immediately when a fault is detected. Because of the skewed manner of inputting the data, if we stop the operation immediately, the checksum property will not hold according to *Theorem 3.4*. Each processor PE_{ij} , $1 \leq i \leq j \leq p$, of the QR tri-array has to take $j - i$ more data and the i^{th} cell of the EDA has to take $p + 1 - i$ more data so that the checksum is satisfied for each row of the systolic array. If each data requires one system clock, we observe that at most p more system clocks are needed to process those unfinished data after the moment a fault is detected. The last row takes one clock and the first row takes p clocks. Generally, the i^{th} row takes $p + 1 - i$ clocks to process the unfinished data. Thus, those rows which take fewer clocks can pipe their final results out to the right to check their checksum while others rows are still working on their unfinished data.

3.3.3 Order-Degraded Reconfiguration

An order-degraded performance is reasonable and often acceptable in many LS applications. A reconfiguration is needed to reroute data paths for order-degraded operation. Many models and approaches can be found in the literatures [58, 44, 57, 92] for the reconfiguration of VLSI array processors. Here we use a similar model described in [58]. When the faulty row, say row k , is determined, the cells in the k^{th} column and row become connection elements and enter a dormant state. In the dormant state, each cell tests itself to check its status repeatedly [58]. The reduced $(p - 1) \times (p - 1)$ tri-array then operates in an order-degraded LS computational manner. Fig. 3.18 shows an example of bypassing the faulty row and the associated column to become an order-degraded LS array. When the (transient) fault is removed, the dormant cells

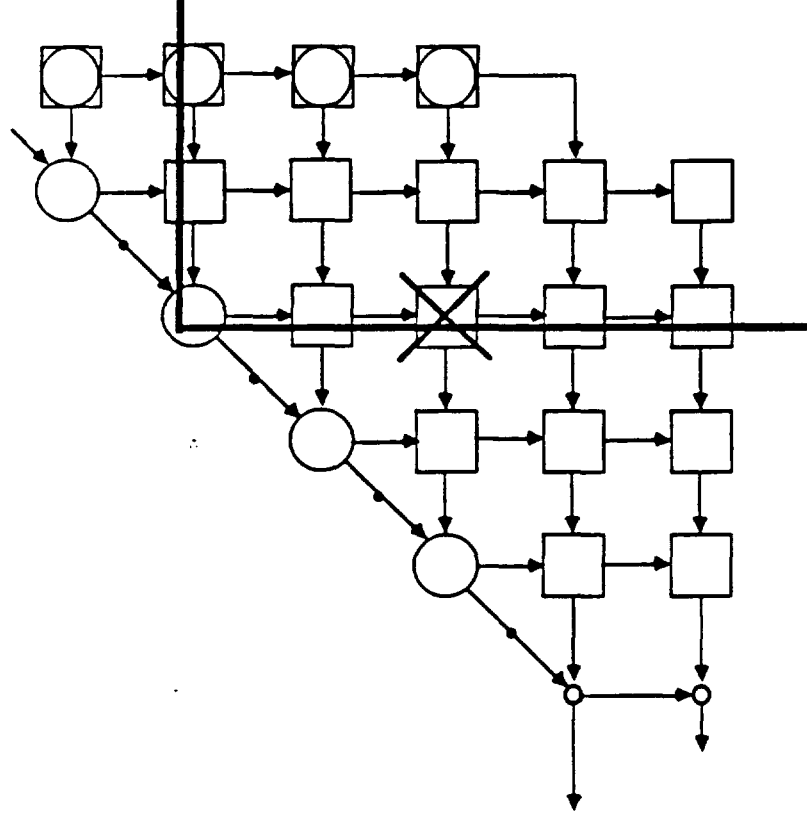


Figure 3.18: Reduced $(p - 1) \times (p - 1)$ tri-array after the deletion of the row and column with a faulty cell.

reactivate and generate an interrupt immediately. The reactivation scheme recovers all of the cells which become connection elements before and turns them into active cells. Then the full-order LS operation is resumed. Detail reviews on various schemes and technologies of reconfiguration can be found in [13].

3.4 Error Propagation and Recovery Latency

In this section, we derive and compare the performances of the proposed fault-tolerant schemes.

3.4.1 Robust Error Detection

Some definitions needed in the analysis are first given. A processor is said to be *faulty* if a fault has occurred in it and is said to have been *contaminated* if it contains erroneous data or states. The *faulty moment* is the time instant a fault first occurs within the processor, and the *contaminating moment* is the time instant the first error occurs in the processor [98]. A fault is said to be *observable* if some of the observable states, such as signal data and system output, of the processor can be affected by the fault. It may take some latent period to change the observable states incorrectly once the fault occurs. An *immediately observable fault* is a fault which affects some of the observable states right after its occurrence. To make the analysis tractable, we first assume all faults are immediately observable faults such that when the fault occurs, it produces error at the output of the faulty processor immediately. Later, we will relax this assumption for more general results. The error is then propagated and contaminates those processors affected. The moment the error is first observed at the output of the system is called the *error observed moment*. First, assume a fault occurs in an internal cell $PE_{ij}, i \neq j$, at a faulty moment. The output of this faulty cell is thus erroneous and can be described by $x'_{out} = x_{out} + \delta$, where x_{out} is the fault-free output and δ is the error generated by the fault. The error propagation path can be described by

$$PE_{ij} \rightarrow PE_{(i+1)j} \rightarrow \cdots \rightarrow PE_{jj},$$

and then $PE_{kl}, k \geq j, l \geq j$ are all contaminated.

From the operations executed by the internal cell, the error is modified to $c_{i+1}\delta$ by $PE_{(i+1)j}$ and the cumulative modifications of the error before reaching the boundary

cell, PE_{jj} , is

$$\eta = \delta \prod_{k=i+1}^{j-1} c_k, \quad (3.63)$$

where c_i is the cosine parameter generated by the boundary cell PE_{ii} . Therefore, c_j and s_j are erroneous and are given by

$$\begin{aligned} c_j &= \frac{\lambda r}{\sqrt{\lambda^2 r^2 + (x_{in} + \eta)^2}}, \\ s_j &= \frac{(x_{in} + \eta)}{\sqrt{\lambda^2 r^2 + (x_{in} + \eta)^2}}. \end{aligned} \quad (3.64)$$

In this case, s_j is no longer proportional to x_{in} , $\underline{a}(j)$ will not be zeroed out by the j^{th} cell of the EDA. The size of the error generated by this cell is

$$\eta_j = c_j x_{in} - s_j \lambda r = -\frac{\lambda r \eta}{\sqrt{r'^2 + 2\eta x_{in} + \eta^2}}, \quad (3.65)$$

where $r' = \sqrt{\lambda^2 r^2 + x_{in}^2}$ is the new updated uncontaminated value of the content of PE_{jj} . Although those c_k 's and s_k 's for $j \leq k \leq p$ are contaminated, $\underline{a}(k)$, $j \leq k \leq p$, are zeroed out by the k^{th} cell of the EDA because of the consistency of the sine and cosine parameters used by the k^{th} column array of the QR tri-array and the EDA. When η_j propagates down to the output of the EDA, η_j is influenced by the contaminated cosines c' of each following row. The error output at e_0 due to an error δ generated at PE_{ij} is then given by

$$\begin{aligned} e_0^\delta(i, j) &= -\gamma \prod_{m=j+1}^p c'_m \frac{\eta \lambda r}{\sqrt{r'^2 + 2\eta x_{in} + \eta^2}} \\ &= \begin{cases} \frac{-\gamma \lambda r \prod_{k=i+1}^{j-1} c_k \prod_{m=j+1}^p c'_m \delta}{\sqrt{r'^2 + 2x_{in} \prod_{k=i+1}^{j-1} c_k \delta + \prod_{k=i+1}^{j-1} c_k^2 \delta^2}} & i \leq j-2, \\ \frac{-\gamma \lambda r \prod_{m=j+1}^p c'_m \delta}{\sqrt{r'^2 + 2x_{in} \delta + \delta^2}} & i = j-1. \end{cases} \end{aligned} \quad (3.66)$$

where $\gamma = \prod_{i=1}^{j-1} c_i \prod_{k=j}^p c'_k$. Next, assume a fault occurs in a boundary cell, PE_{jj} , $1 \leq j \leq p$, at the faulty moment. Both erroneous c'_j and s'_j produced by PE_{jj} can be

written by

$$c'_j = \frac{\lambda r + \delta_c}{r'_\epsilon}, \quad s'_j = \frac{x_{in} + \delta_s}{r'_\epsilon}, \quad (3.67)$$

where δ_c and δ_s represent errors in the numerators while r'_ϵ represents the erroneous content of the denominators of c_j and s_j . The error produced by the j^{th} cell of the EDA is then given by

$$\eta_j = c'_j x_{in} - s'_j \lambda r = \frac{(x_{in} \delta_c - \lambda r \delta_s)}{r'_\epsilon}, \quad (3.68)$$

and the output error at e_0 due to a faulty boundary cell is given by

$$e_0^\delta(j, j) = \gamma \prod_{m=j+1}^p c'_m \cdot \frac{x_{in} \delta_c - \lambda r \delta_s}{r'_\epsilon}. \quad (3.69)$$

A fault can occur either in the internal or boundary cell. First assume the fault occurs in an internal cell. The cosine parameter, c_j , produced by PE_{jj} is bounded by $0 \leq c_j \leq 1$.

Lemma 3.3 *For some c_j , if there exists an $n \in I$ such that $c_j(n) \neq 0$, then $c_j(m) > 0$ for all $m > n$, $m \in I$.*

Proof: The cosine parameter is given by $c_{k+1} = \lambda r(k)/r'(k)$, where $r(k+1) = r'(k) = \sqrt{\lambda^2 r^2(k) + x_{in}^2(k)}$. $\exists n \rightarrow c_j(n) \neq 0$ is equivalent to say that $\exists n \rightarrow r(n) \neq 0$ or $r(n) > 0$. Since $r(k+1) = r'(k) \geq \lambda r(k)$, we have $r(k) > 0$ for all $k \geq n$. Therefore, $c_j(k) > 0$ for all $k > n$. \square

For linearly independent input column vectors, all $c_j(n) \neq 0$, and from (3.66), we see $e_0^\delta \neq 0$ if an error has been introduced. Thus the fault can always be detected in this case. When the fault occurs in a boundary cell, from (3.69) we conclude that the only chance of a loss of detection is that $e_0^\delta = 0$. This happens only if $\delta_c/\delta_s = \lambda r/x_{in}$, which is very unlikely since the value of δ_c , δ_s , r , and x_{in} are not related in any

manner. Furthermore, even if δ_c/δ_s equals $\lambda r(n)/x_{in}(n)$ for some instant of time at n , it is even more unlikely that δ_c/δ_s equal $\lambda r(n+1)/x_{in}(n+1)$ at time $n+1$. We can summarize these results in the following theorem. Note that the statement of *Theorem 3.5* is valid only for infinite-precision implementation.

Theorem 3.5 (Robust Error Detection Theorem) *An error produced by a faulty processor at the faulty moment will be detected at the EDA output e_0 and the probability of error detection given a fault occurs equal one. That is,*

$$Pr(\text{error detected at } e_0 \mid \text{a fault occurred}) = 1. \square \quad (3.70)$$

3.4.2 Latencies

Now, we consider some basic issues related to latencies in the array.

Definition 5.1: The *system latency*, t_s , is the time between the moment of data input to the system and the moment of the output of this data from the system.

Definition 5.2: The *processing latency*, t_p , of processor PE_{ij} is the time between the moment a data in a wavefront inputs to the system and the moment PE_{ij} is processing data from that wavefront.

Definition 5.3: The *error propagation latency*, t_e , is the time between the faulty moment and the error observed moment. It is clear that the system latency of the QR recursive LS array depends on the number of processors and delay elements on the boundary and is given by $t_s = 2p + 1$. The processing latency of processor PE_{ij} , $1 \leq i \leq j \leq p + 1$, is given by $t_p = (i + j) - 1$. Since there are a totally of $p(p + 3)/2$ processors, the expected processing latency is

$$E(t_p) = \frac{2}{p(p + 3)} \sum_{i=1}^p \sum_{j=i}^{p+1} (i + j - 1) = \frac{p^2 + 4p + 1}{p + 3}. \quad (3.71)$$

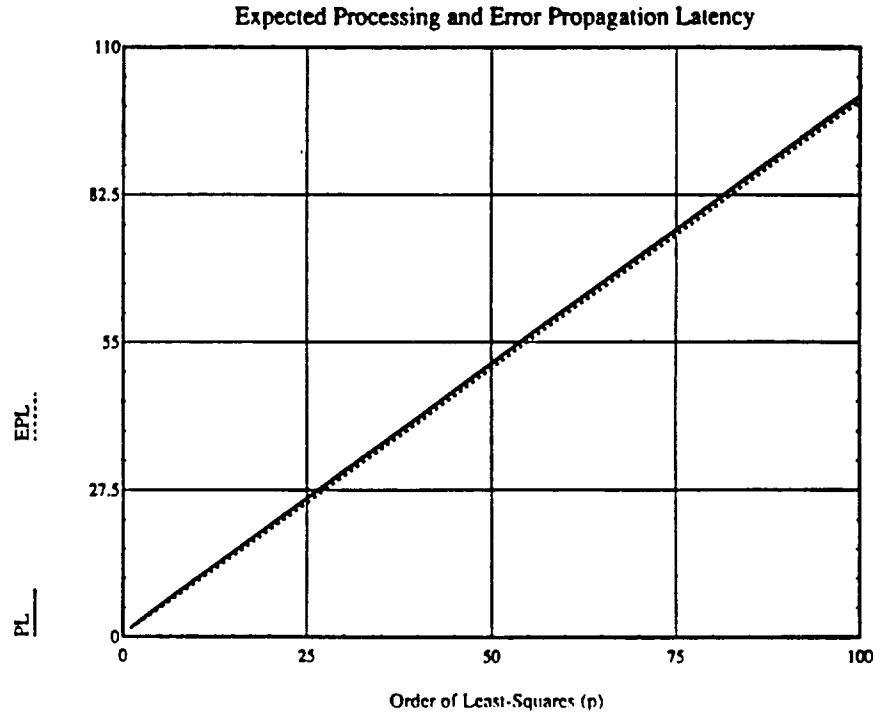


Figure 3.19: Plot of expected processing latency and expected error propagation latency versus the order of the LS estimation.

The error propagation latency is given by

$$t_e = t_s - t_p = 2(p + 1) - (i + j), \quad (3.72)$$

and the expected value is

$$E(t_e) = \frac{(p + 1)(p + 2)}{(p + 3)}. \quad (3.73)$$

Fig. 3.19 shows the plots of $E(t_p)$ and $E(t_e)$.

Definition 5.4: The *fault diagnosis time*, t_f , of a faulty processor PE_{ij} is the minimum time required to locate the faulty row right after the error observed moment.

Definition 5.5: The *recovery latency*, t_r , be the time between the faulty moment and the moment the faulty row is determined. Since the system latency is $2p + 1$, for the FFL method the fault diagnosis time of processor PE_{ij} for the array is

$t_f^{FFL} = (2p+1) + i$. We can show that the fault diagnosis time for the CSE method is $t_f^{CSE} = p+2 - i$. The expected value for fault diagnosis time are $E(t_f^{FFL}) = (5/2)p+1$ and $E(t_f^{CSE}) = (1/2)p + 2$ respectively. By the definition of the recovery latency, we have $t_r = t_e + t_f$. Therefore, the recovery latency are $t_r^{FFL} = 4p - j + 3$ and $t_r^{CSE} = 3p - 2i - j + 4$, while the expected recovery latency are

$$\begin{aligned} E(t_r^{FFL}) &= \frac{7p^2 + 23p + 10}{2p + 6}, \\ E(t_r^{CSE}) &= \frac{3p^2 + 13p + 16}{2p + 6} \end{aligned} \quad (3.74)$$

respectively. Due to the facts that multiple ports can be accessed externally and we can use the parallel pipe-out feature of the CSE method, it is not surprised that the performance of the CSE method is better than that of the FFL method as indicated in Fig. 3.20. However, for both cases, the order of the expected recovery latency is $O(p)$, which is linear with respect to p . In practice, a (transcient) fault may not be necessarily an immediately observable fault. Without this assumption, all the values obtained in this section become the lower bounds of those parameters. That is, performance obtained by the assumption of immediately observable fault is the best performance we can achieve.

3.5 Conservation Test

Thus far the fault-tolerance system discussed above is designed to detect the fault occurring in the QR triarray and the EDA. It is not applicable to the RA which computes the real desired response. In this section, we introduce some conservation properties to tackle this problem and the data is still assumed to be real.

A: Cell Level Test

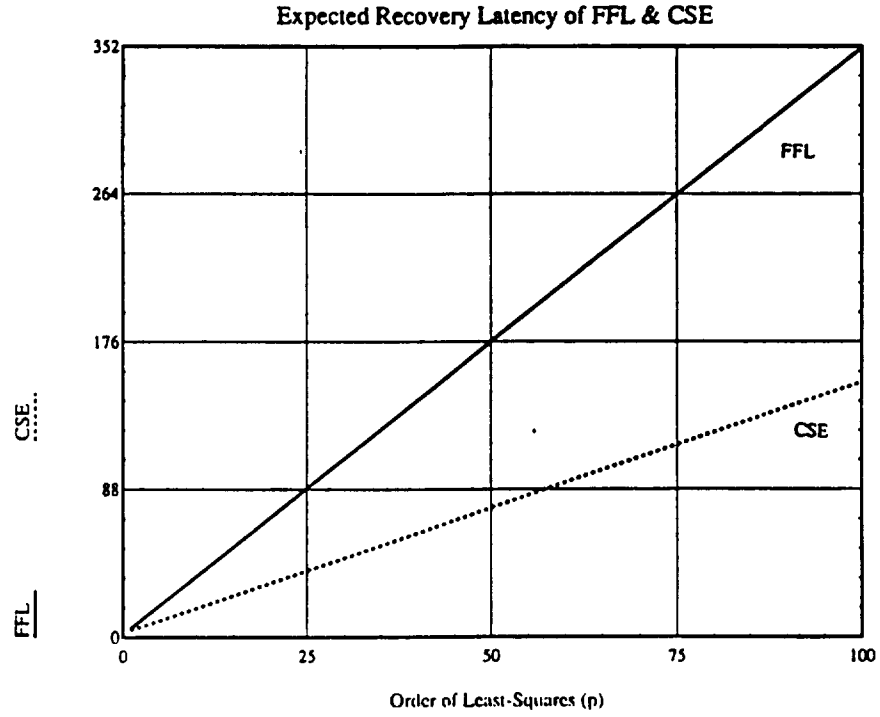


Figure 3.20: Comparisons of expected recovery latency for FFL and CSE methods.

A1: Energy Conservation The rotation operation of internal cell is described in Fig.1. Denote the updated r as r' . Then it can be seen that

$$x_{out}^2 + r'^2 = x_{in}^2 + \lambda^2 r^2, \quad (3.75)$$

which means the energy, (*i.e.* the 2-norm) is conserved before and after the operation. This is due to the fact that the Givens rotation transformation is an unitary transformation which conserves the energy.

A2: Inverse of Unitary Matrix The above energy conservation test requires the square operation which is different from the operation of the internal cell. In a VLSI/WSI system, it is desirable to keep the number of different kinds of processors or cells as low as possible [74, 54]. Observe that the inverse of a unitary

matrix is the Hermitian transpose of it. If there is no fault occurring during the computations, the inverse computation should give the original values. That is

$$\begin{bmatrix} x_{in} \\ \lambda r \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x_{out} \\ r' \end{bmatrix}. \quad (3.76)$$

One of the above inverse computation is sufficient to detect a fault. For example, the inverse computation of $x_{in} = cx_{out} + sr'$ will not match the original x_{in} , if x_{out} or r' is erroneous. This kind of test can be carried out by the internal cell.

B: System Level Test

Consider the RA which computes the real desired response. Based on the previous discussion, the energy is conserved for unitary transformation. Denote $P_i(n)$, $1 \leq i \leq p$, to be the content of the i^{th} cell of the array at time n and d as defined in section 2. The system level energy conservation is described by

$$\sum_{i=1}^p P_i^2(n+i-1) = \sum_{j=1}^n d^2(j). \quad (3.77)$$

That is, the energy of the input signal is equal to the energy in the RA. Note that the introduced time index is used to describe the operations of continuously updating the incoming signal. Both cell and system level tests can be built either in the system or applied externally depending on the requirements of the application.

Chapter 4

Dynamic Range, Stability, and Fault-tolerant Capability of Finite-precision QRD RLS Systolic Algorithm

In this chapter, we first observe that the cosine parameters generated by boundary cells will eventually reach *quasi steady-state* if λ is close to one which is the usual case. We will show that the quasi steady-state and ensemble values of sine and cosine parameters are the same for all boundary cells. It is independent of the statistics of the input data sequence and the position of the boundary cell which generates the sine and cosine parameters. Simulation results are presented to support this observation. These results yield the tools needed to further investigate many properties of the QRD LS systolic algorithm. Then, we can obtain upper bounds of the dynamic range of processing cells. Thus, lower bounds on the memory size can be obtained

from upper bounds of the dynamic range to prevent overflow and to ensure correct operations of the QRD LS algorithm. With these results, we reconsider the stability problem under quantization effects with a more general analysis and obtain tighter bounds than given in previous work [60]. Two important factors of the fault-tolerant capability, the *missing error detection* and the *false alarm* effects are also studied in this chapter.

Quasi steady-state of the rotation parameters is discussed in section 4.1. Dynamic range and lower bound on memory size are derived in section 4.2. Stability and quantization effects are studied in section 4.3. Finally, the fault-tolerant capability is presented in section 4.4.

4.1 Quasi Steady-State and Ensemble Behavior

From the updated recursive equation of the boundary cell (Fig. 2.2), we have

$$r^2(k+1) = \lambda^2 r^2(k) + x^2(k) = \sum_{i=0}^k \lambda^{2i} x^2(k-i). \quad (4.78)$$

Assume $\{x\}$ is a bounded random sequence of zero-mean and variance σ^2 , then the expected value of $r^2(k+1)$ is

$$E[r^2(k+1)] = \sum_{i=0}^k \lambda^{2i} E(x^2(k-i)) = \sigma^2 \frac{1 - \lambda^{2(k+1)}}{1 - \lambda^2}. \quad (4.79)$$

For $|\lambda| < 1$, then

$$\lim_{k \rightarrow \infty} E[r^2(k)] = \frac{\sigma^2}{1 - \lambda^2}. \quad (4.80)$$

Since $\sqrt{\cdot}$ is a concave function, from Jensen inequality

$$\lim_{k \rightarrow \infty} E(r(k)) \leq \lim_{k \rightarrow \infty} \sqrt{E[r^2(k)]} = \frac{\sigma}{\sqrt{1 - \lambda^2}}, \quad (4.81)$$

and from (4.78)

$$\frac{|x_{min}|}{\sqrt{1-\lambda^2}} \leq \lim_{k \rightarrow \infty} r(k+1) \leq \frac{|x_{max}|}{\sqrt{1-\lambda^2}} \quad (4.82)$$

where $|x_{max}|$ and $|x_{min}|$ are the maximum and minimum values of $\{|x|\}$, respectively.

The cosine parameter of the Givens rotation is computed by $c(k+1) = \lambda r(k)/r(k+1)$. The steady-state of this parameter exists if $\lim_{k \rightarrow \infty} c(k)$ exists. For the sequence $\{c(\cdot)\}$ to have a steady-state, we need $\lim_{k \rightarrow \infty} r(k)/r(k+1) = \alpha$, where α is a constant. If $\alpha < 1$, then the sequence $\{r(\cdot)\}$ is unbounded which conflicts with (4.82) that indicates $\{r(\cdot)\}$ should be bounded; if $\alpha > 1$, then $\lim_{k \rightarrow \infty} r(k) = 0$ which, again, conflicts with (4.82) (unless $\{x\}$ is a zero sequence). Therefore, α has to be a unity to guarantee the steady-state of $\{c(\cdot)\}$ exists. That is,

$$\lim_{k \rightarrow \infty} \frac{r(k)}{r(k+1)} \rightarrow 1, \quad (4.83)$$

and the steady-state value of cosine, if exists, is

$$\lim_{k \rightarrow \infty} c(k+1) = \lim_{k \rightarrow \infty} \frac{\lambda r(k)}{r(k+1)} = \lambda. \quad (4.84)$$

From (4.78), we can see that if $\lambda = 1$, then $\lim_{k \rightarrow \infty} r(k) \rightarrow \infty$ such that $\lim_{k \rightarrow \infty} r(k)/r(k+1) = 1$. In this case, though the steady-state of $\{c(\cdot)\}$ exists, $\{r(\cdot)\}$ is unbounded. Usually λ is chosen between .99 and 1 which is very close to one¹. When we update $r(k)$ to $r(k+1)$ using (4.78), a λ portion of $r(k)$ is forgotten and an input $x(k)$ is added into it. If λ is close to one, when k is very large, $r(k)$ will come close to $r(k+1)$ and the input $x(k)$ plays less and less significant role in computing $r(k+1)$ as the case when $\lambda = 1$. It is obvious that for λ close to one,

$$\lim_{k \rightarrow \infty} Er(k) \approx \lim_{k \rightarrow \infty} Er(k+1).$$

¹For different expressions as in [32, 60, 78], λ is between .98 and 1

Therefore, from the averaging principle [76] which has been used successfully in many situations, the expected cosine can be approximated by

$$\lim_{k \rightarrow \infty} Ec(k+1) \simeq \lambda \frac{Er(k)}{Er(k+1)} \approx \lambda. \quad (4.85)$$

When λ is close to one, from above discussions, we have

$$\lim_{k \rightarrow \infty} c(k) = \lim_{k \rightarrow \infty} \frac{\lambda r(k)}{r(k+1)} = \lambda + \delta(\lambda, x), \quad (4.86)$$

where $\delta(\lambda, x)$ represents the small deviation due to the forgotten λ portion of r and input of x . If δ is very small such that it is negligible when k is large, we say that the sequence $\{c(\cdot)\}$ reaches the *quasi steady-state*.

Generally, it is difficult to quantitatively describe $\delta(\lambda, x)$. Here we model the input signal to the systolic array as a second-order AR process described by

$$x(n) + a_1 x(n-1) + a_2 x(n-2) = v(n), \quad (4.87)$$

where $v(n)$ is a white Gaussian noise process of zero mean and unit variance. By choosing of different AR parameters a_1 and a_2 , we obtain different realizations of the AR process [32]. In our simulations, three different categories of signal are encountered. The first category consists of three stationary AR processes which are AR1 ($a_1 = -0.1, a_2 = -0.8$), AR2 ($a_1 = 0.1, a_2 = -0.8$) with real roots and AR3 ($a_1 = -0.975, a_2 = 0.95$) with complex-conjugate roots. The second category is a non-stationary AR process, AR4 ($a_1 = -0.6, a_2 = -0.5$), and the third category is a white Gaussian noise process, WN, with zero mean and unit variance. All of the AR processes are normalized to unit variance. Table 4.3 shows the mean distribution for different input data with different λ values. This table justifies the result in (4.85). Table 4.4 shows the variance distribution of different input data with different λ

	AR1	AR2	AR3	AR4	WN
$\lambda = .980$.9800	.9800	.9802	.9799	.9801
$\lambda = .985$.9849	.9849	.9851	.9848	.9850
$\lambda = .990$.9897	.9897	.9900	.9897	.9899
$\lambda = .991$.9907	.9907	.9910	.9907	.9909
$\lambda = .993$.9927	.9927	.9930	.9927	.9929
$\lambda = .995$.9947	.9947	.9950	.9947	.9949
$\lambda = .997$.9967	.9967	.9970	.9967	.9969
$\lambda = .999$.9985	.9985	.9987	.9985	.9986

Table 4.3: Mean distribution for different input data with different λ values

value. The values of those variances are in the order of 10^{-4} to 10^{-6} which implies that δ is indeed very small. They can be closely approximated by using quadratic polynomials as follows,

$$\begin{aligned}
AR1 : \quad \sigma_c^2(\lambda) &= 1.5938 - 3.182\lambda + 1.5882\lambda^2 \\
AR2 : \quad \sigma_c^2(\lambda) &= 1.5991 - 3.1919\lambda + 1.5928\lambda^2 \\
AR3 : \quad \sigma_c^2(\lambda) &= 1.5812 - 3.1595\lambda + 1.5784\lambda^2 \\
AR4 : \quad \sigma_c^2(\lambda) &= 1.4492 - 2.8936\lambda + 1.4444\lambda^2 \\
AR5 : \quad \sigma_c^2(\lambda) &= 1.6437 - 3.2904\lambda + 1.6431\lambda^2
\end{aligned} \tag{4.88}$$

We can see, although the statistics of the input data are different, the variances can be described by λ in a very similar way (see Fig. 4.21). This means, when λ is close to one and the quasi steady-state is reached, the size of the variation δ is mainly governed by λ instead of the statistics of the input data. Fig. 4.21 shows the plots of the variances in dB scale.

	AR1	AR2	AR3	AR4	WN
$\lambda = .980$	7.3885e-4	7.5465e-4	6.8163e-4	6.6721e-4	7.3367e-4
$\lambda = .985$	4.3970e-4	4.5144e-4	3.9577e-4	3.9517e-4	4.3308e-4
$\lambda = .990$	2.0903e-4	2.1463e-4	1.8376e-4	1.8918e-4	2.0080e-4
$\lambda = .991$	1.7154e-4	1.7875e-4	1.4883e-4	1.5562e-4	1.6659e-4
$\lambda = .993$	1.0991e-4	1.1390e-4	9.1016e-5	9.6440e-5	1.0323e-4
$\lambda = .995$	5.9724e-5	6.0796e-5	4.6789e-5	5.1856e-5	5.3525e-5
$\lambda = .997$	2.3007e-5	2.4735e-5	1.6808e-5	1.9908e-5	2.0504e-5
$\lambda = .999$	4.1127e-6	3.1590e-6	3.5167e-6	4.3511e-6	4.6490e-6

Table 4.4: Variance distributions for different input data with different λ values

With these results, we conclude that sequence $\{c(\cdot)\}$ reaches the quasi steady-state regardless the input statistics if λ is close to one. Thus, we can write

$$\begin{aligned}
\lim_{k \rightarrow \infty} c(k+1) &\simeq \lim_{k \rightarrow \infty} Ec(k+1) \simeq \lambda, \\
\lim_{k \rightarrow \infty} s(k+1) &\simeq \lim_{k \rightarrow \infty} Es(k+1) \simeq \sqrt{1 - \lambda^2}.
\end{aligned} \tag{4.89}$$

The quasi steady-state and ensemble values of sine and cosine parameters are the same for all boundary cells. It is independent of the statistics of the input data sequence and the position of the boundary cell which generates the sine and cosine parameters. These results yield the tools needed to further investigate many properties of the QRD LS systolic algorithm.

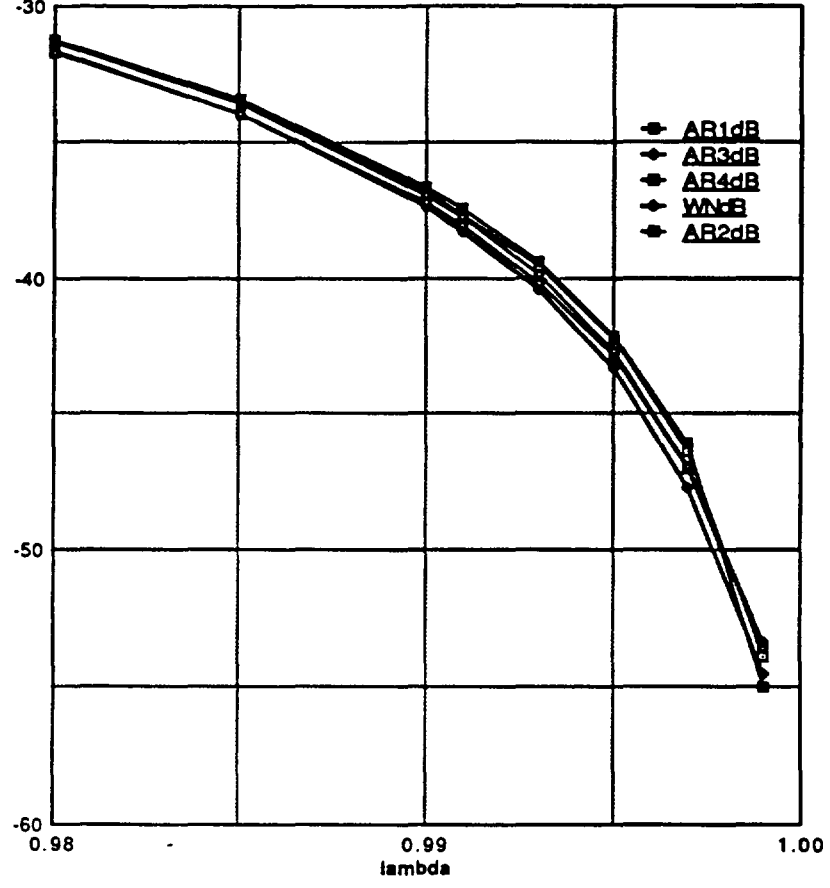


Figure 4.21: Plots of the variances in dB

4.2 Dynamic Range and Lower Bound on Memory Size

The dynamic range of the content of boundary cell PE_{11} can be upper bounded by

$$\lim_{k \rightarrow \infty} r_{11}^2(k+1) = \lim_{k \rightarrow \infty} \sum_{i=0}^k \lambda^{2i} x^2(k-i) \leq \lim_{k \rightarrow \infty} x_{max}^2 \sum_{i=0}^k \lambda^{2i} = \frac{x_{max}^2}{1-\lambda^2}. \quad (4.90)$$

Therefore,

$$\lim_{k \rightarrow \infty} |r_{11}(k)| \leq \frac{|x_{max}|}{\sqrt{1-\lambda^2}} \triangleq \mathfrak{R}. \quad (4.91)$$

For internal cell PE_{1j} , we have

$$|r_{1j}(k+1)| = |s(k)x(k) + c(k)\lambda r(k)|$$

$$\begin{aligned}
&= |s(k)x(k) + c(k)\lambda[s(k-1)x(k-1) + c(k-1)\lambda r(k-1)]| \\
&\leq \sum_{i=0}^k \lambda^i |x(k-i)s(k-i)| \prod_{j=0}^{i-1} c(k-j) \\
&\leq |x_{max}| \sum_{i=0}^k \lambda^i |s(k-i)| \prod_{j=0}^{i-1} c(k-j)
\end{aligned} \tag{4.92}$$

From the basic relation between the geometric mean and the arithmetic mean, we know

$$\left(\frac{a_1 + a_2 + \cdots + a_n}{n}\right)^n \geq a_1 \cdot a_2 \cdots a_n. \tag{4.93}$$

If n is large enough, then from the law of large number, we know

$$\lim_{n \rightarrow \infty} \frac{a_1 + a_2 + \cdots + a_n}{n} \rightarrow E(a).$$

Therefore,

$$E(a)^n \geq \prod_{i=1}^n a_i,$$

when n is large. We can further simplify the bound for $k \rightarrow \infty$ by using this inequality as follows,

$$\begin{aligned}
\lim_{k \rightarrow \infty} |r_{1j}(k+1)| &\leq |x_{max}| \lim_{k \rightarrow \infty} \sum_{i=0}^k \lambda^i s(k-i) E(c(k-i))^i \\
&= |x_{max}| \sum_{i=0}^k \lambda^{2i} \cdot \sqrt{1-\lambda^2} = \frac{|x_{max}|}{\sqrt{1-\lambda^2}} = \mathfrak{R}.
\end{aligned} \tag{4.94}$$

From (4.90) and (4.93), we can see the steady-state dynamic range of the first row is upper bounded by \mathfrak{R} for both boundary and internal cells. The dynamic range of the second row depends on the output of internal cells of the first row. Denote the output of the first row as x_{out} , we have

$$x_{out}(k+1) = c(k)x(k) - s(k)\lambda r(k). \tag{4.95}$$

The first term of the right-hand side of (4.95) can be bounded by

$$\lim_{k \rightarrow \infty} |c(k)x(k)| \leq \lambda |x_{max}| \tag{4.96}$$

and from (4.94) the second term is bounded by

$$\lim_{k \rightarrow \infty} |s(k)\lambda r(k)| \leq \sqrt{1 - \lambda^2} \cdot \lambda \frac{|x_{max}|}{\sqrt{1 - \lambda^2}} = \lambda |x_{max}|. \quad (4.97)$$

There are two possible cases.

Case 1: Highly fluctuated input

The value of $x(k)$ may vary differently from time to time such that $s(k)$ may have an opposite sign of $x(k)$. For this case

$$\lim_{k \rightarrow \infty} |x_{out}(k)| \leq 2\lambda |x_{max}|. \quad (4.98)$$

Case 2: Smooth input

For this case, the input data sequence does not change its value rapidly, therefore $s(k)$ may always have the same sign as $x(k)$. The bound is

$$\lim_{k \rightarrow \infty} |x_{out}(k)| \leq \lambda |x_{max}|. \quad (4.99)$$

From (4.91) and (4.94), it is obvious the steady-state dynamic range of the second row is bounded by

$$\lim_{k \rightarrow \infty} |r_{2j}(k)| \leq \frac{\lambda |x_{max}|}{\sqrt{1 - \lambda^2}} = 2\lambda \mathfrak{R}, \quad (4.100)$$

for the highly fluctuated input and

$$\lim_{k \rightarrow \infty} |r_{2j}(k)| \leq \lambda \mathfrak{R}, \quad (4.101)$$

for the smooth input. From above results, the steady-state dynamic range of the m^{th} row is bounded by

$$\lim_{k \rightarrow \infty} |r_{mj}(k)| \leq (2\lambda)^{m-1} \cdot \mathfrak{R}, \quad (4.102)$$

for the highly fluctuated input and

$$\lim_{k \rightarrow \infty} |r_{mj}(k)| \leq (\lambda)^{m-1} \mathfrak{R}, \quad (4.103)$$

for the smooth input.

Fig. 4.22 shows a simulation of the contents of internal cells of the first row and the second row. Denote B_m as the word-length of the m^{th} row, to prevent overflow and to insure the correct operation of the QRD LS algorithm, we require $2^{B_m} \geq (2\lambda)^{m-1} \mathfrak{R}$ for fixed point operation, and therefore

$$B_m \geq \lceil (m-1)(1 + \log_2 \lambda) + \log_2 \mathfrak{R} \rceil. \quad (4.104)$$

The memory size required for each row to prevent overflow increases and decreases for the highly fluctuated and smooth inputs respectively. For the fluctuated input, when $(2\lambda)^{m-1} = 2$, one more bit is needed for the memory of each successive row. The number of rows m for each increase is

$$m = \lceil 1 + \frac{1}{1 + \log_2 \lambda} \rceil, \quad (4.105)$$

which is a monotonically decreasing function of λ . If $\lambda \leq 0.5$, then there is no such m exists. That is, the memory size of the array can be fixed at \mathfrak{R} without the overflow problem. For smooth input, when $\lambda^{m-1} = \frac{1}{2}$, one bit can be discarded from the memory of the following rows. The number of rows m for each decrease is

$$m = \lceil 1 - \frac{1}{\log_2 \lambda} \rceil, \quad (4.106)$$

which is a monotonically increasing function of λ . For $\lambda \leq 0.5$, $m = 2$. That is, for every two rows we can decrease one bit for the memory.

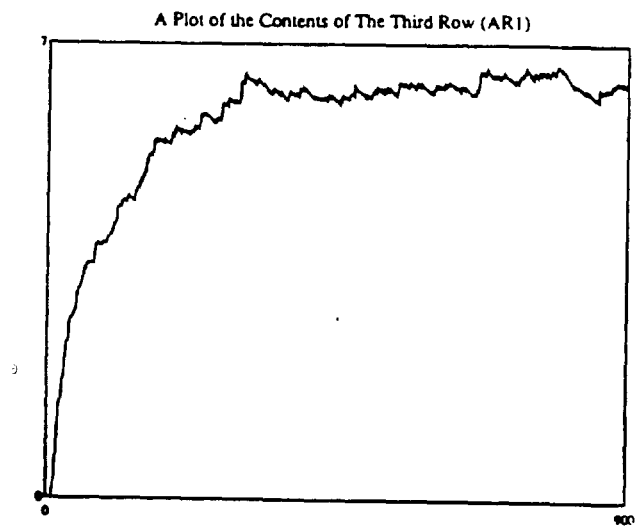
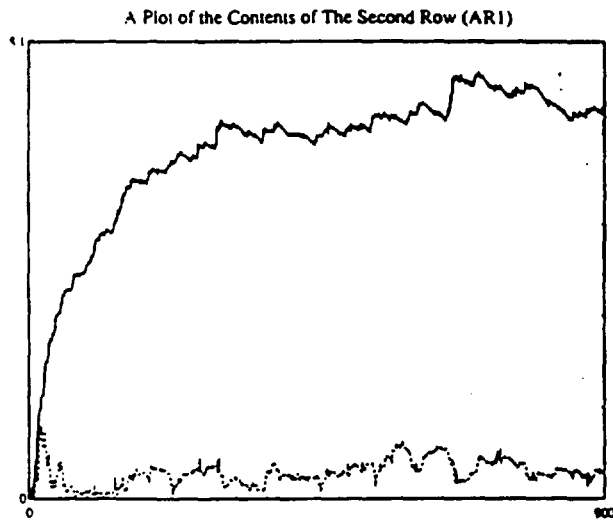
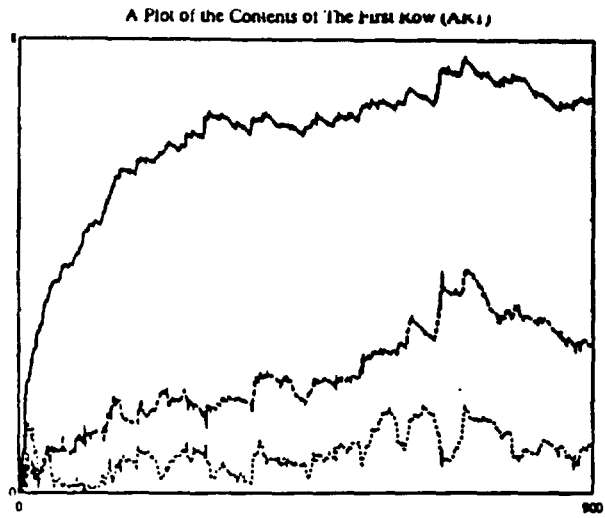
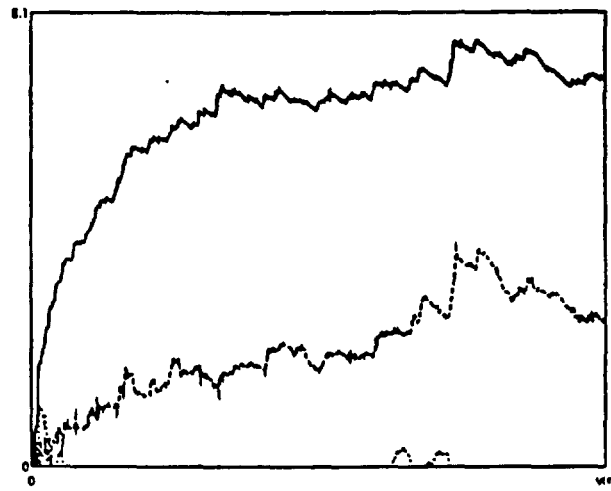
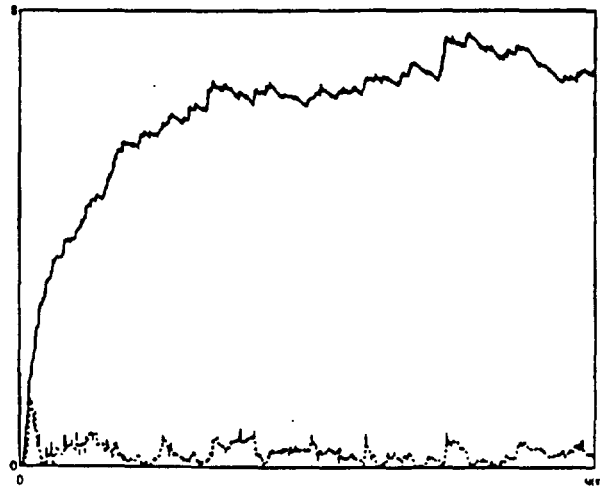


Figure 4.22: Contents of the Cells

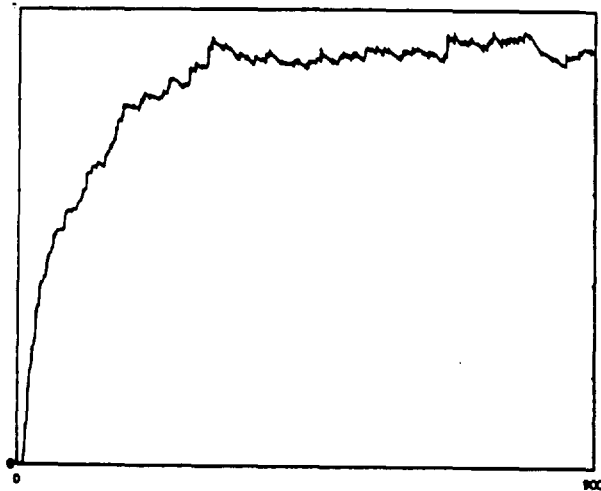
A Plot of the Contents of The First Row (AR2)



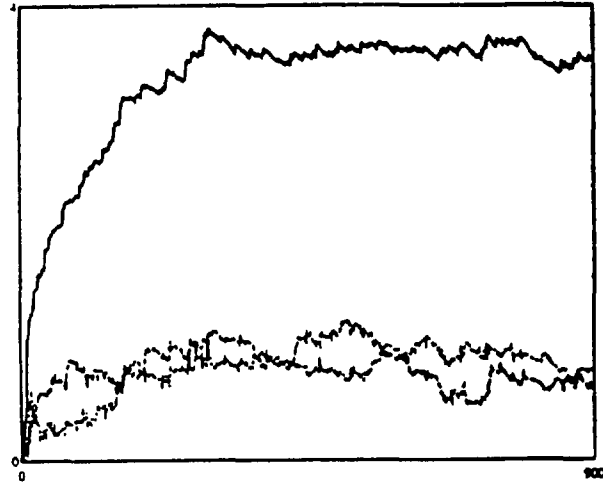
A Plot of the Contents of The Second Row (AR2)



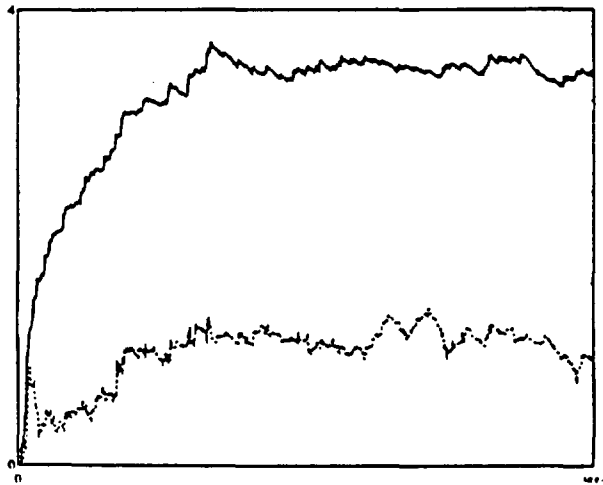
A Plot of the Contents of The Third Row (AR2)



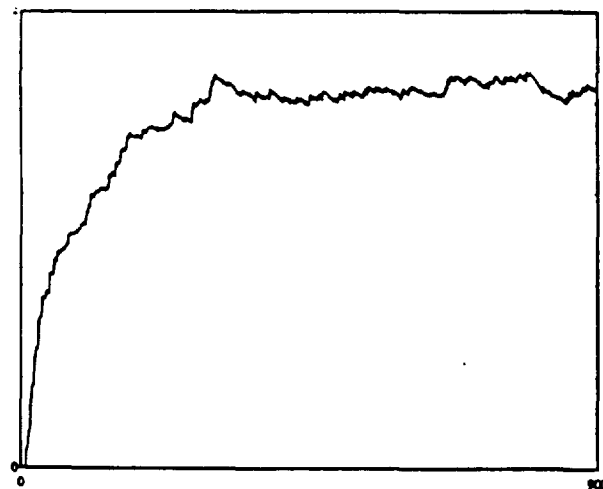
A Plot of the Contents of The First Row (AR3)



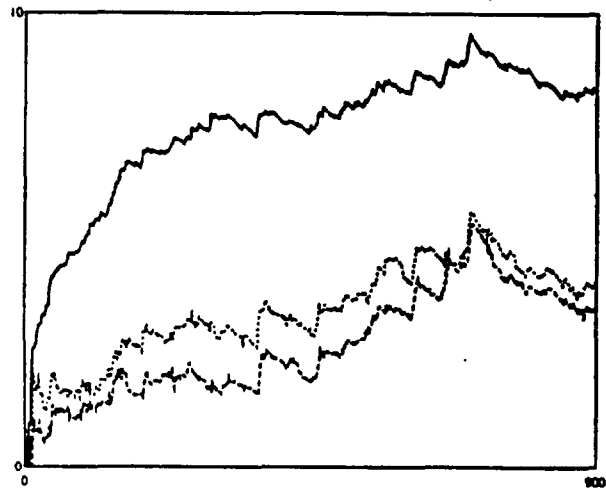
A Plot of the Contents of The Second Row (AR3)



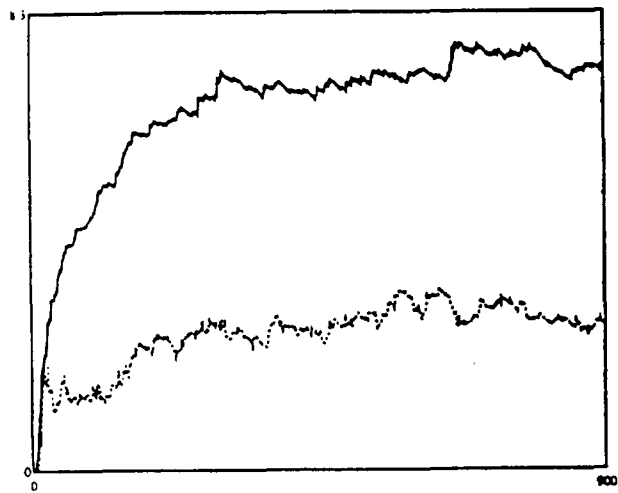
A Plot of the Contents of The Third Row (AR3)



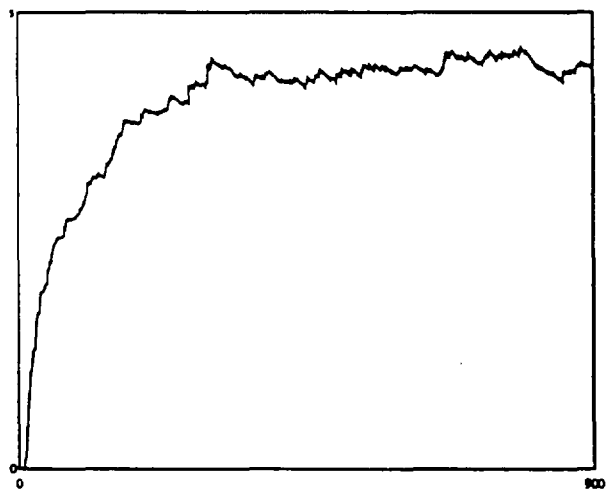
A Plot of the Contents of The First Row (AR4)



A Plot of the Contents of The Second Row (AR4)



A Plot of the Contents of The Third Row (AR4)



4.3 Stability and Quantization Effect

In this section, we consider the stability under quantization effects. Here, the stability is defined in the sense of bounded input/bounded output (BIBO) as in [60]. From (4.98) and (4.99), the output of the m^{th} row is bounded by

$$\lim_{k \rightarrow \infty} |x_{out_m}| \leq (2\lambda)^{m-1} |x_{max}|, \quad (4.107)$$

for the highly fluctuated input and

$$\lim_{k \rightarrow \infty} |x_{out_m}| \leq \lambda^{m-1} |x_{max}| \quad (4.108)$$

for the smooth input.

The order p of the LS problem is finite and fixed. The output of the last row of the QR triarray is bounded, in the worst case, by $\lim_{k \rightarrow \infty} |x_{out_p}| \leq (2\lambda)^{p-1} |x_{max}|$. The residual is then asymptotically bounded by

$$\lim_{k \rightarrow \infty} |e(k)| = \lim_{k \rightarrow \infty} \gamma(k) |x_{out_p}(k)| \leq (2\lambda)^{p-1} |x_{max}|, \quad (4.109)$$

where $\gamma(k) = \prod_{i=1}^p c_i(k)$ [78]. Thus, for $\lambda < 1$, if the input data are bounded, that is, $|x_{max}| < \infty$, the output is always bounded. The QRD LS systolic array constitutes a BIBO stable system under unlimited precision condition. In practice, the memory size of each processing cell is finite-length. Leung and Haykin [60] first considered the stability under this effect and showed the QRD LS algorithm is stable under finite-precision implementation. Here we reconsider this problem and obtain a more general analysis and a tighter bound.

Let $Q(\cdot)$ be the quantization operator and \tilde{x} be the quantized value of x . Since the quantization error for the additions of quantized parameters is much smaller than

that of the multiplications of them, to make the analysis simpler, we may express the quantization error for additions as

$$Q\left(\sum_{i=1}^n \tilde{a}_i\right) = \sum_{i=1}^n \tilde{a}_i + \delta_n \quad (4.110)$$

From (4.78), the square of the quantized content of the boundary cell is

$$\tilde{r}^2(k+1) = Q(Q(\tilde{\lambda}^2 \tilde{r}^2(k)) + Q(\tilde{x}^2(k))) = \sum_{i=0}^k Q(\tilde{\lambda}^{2i} \tilde{x}^2(k-i)) + \delta_{k+1}. \quad (4.111)$$

Since the quantization operator Q is a bounded operator such that $|Q(x)| \leq K|x|$ for all x and some K [60], (4.111) can be bounded by

$$\begin{aligned} |\tilde{r}^2(k+1)| &\leq K_0 |\tilde{\lambda}^{2k} \tilde{x}^2(0)| + K_1 |\tilde{\lambda}^{2(k-1)} \tilde{x}^2(1)| + \cdots + K_k |\tilde{x}^2(k)| + \delta_{k+1} \\ &\leq K_{max} \cdot \tilde{x}_{max}^2 (1 + \tilde{\lambda}^2 + \cdots + \tilde{\lambda}^{2k}), \end{aligned} \quad (4.112)$$

where \tilde{x}_{max} is the maximum quantized value of sequence \tilde{x} . The asymptotic behavior can be obtained by taking the limit on both sides. Thus,

$$\lim_{k \rightarrow \infty} |\tilde{r}^2(k)| \leq K_{max} \cdot \tilde{x}_{max}^2 \frac{1}{1 - \tilde{\lambda}^2}. \quad (4.113)$$

Therefore, the quantized content is

$$\begin{aligned} \lim_{k \rightarrow \infty} |\tilde{r}(k)| &= \lim_{k \rightarrow \infty} Q(\sqrt{\tilde{r}^2(k)}) \\ &\leq K'_{max} \frac{|\tilde{x}_{max}|}{\sqrt{1 - \tilde{\lambda}^2}} \triangleq K'_{max} \tilde{\mathfrak{R}}. \end{aligned} \quad (4.114)$$

With the same arguments as in section 4.1, we then have

$$\lim_{k \rightarrow \infty} \frac{\tilde{r}(k)}{\tilde{r}(k+1)} \rightarrow 1 \quad (4.115)$$

and the quantized steady-state value of cosine is

$$\lim_{k \rightarrow \infty} \tilde{c}(k+1) = \lim_{k \rightarrow \infty} \frac{\tilde{\lambda} \tilde{r}(k)}{\tilde{r}(k+1)} = \tilde{\lambda}, \quad (4.116)$$

and the quantized steady-state value of sine is

$$\lim_{k \rightarrow \infty} \check{s}(k+1) = \sqrt{1 - \check{\lambda}^2}.$$

Analogous to section 4.1, we can further obtain $\lim_{k \rightarrow \infty} E\check{c}(k) = \check{\lambda}$ and $\lim_{k \rightarrow \infty} E\check{s}(k) = \sqrt{1 - \check{\lambda}^2}$.

Now consider the quantized content of the internal cell, from (4.92)

$$\begin{aligned} |\check{r}_{1j}(k+1)| &= |Q(Q(\check{s}(k)\check{x}(k)) + Q(\check{c}(k)\check{\lambda}\check{r}(k)))| \\ &= \sum_{i=0}^k Q(\check{\lambda}^i |\check{x}(k-i)\check{s}(k-i)| \prod_{j=0}^{i-1} \check{c}(k-j)) + \delta_{k+1} \\ &\leq K''_{max} |\check{x}_{max}| \sum_{i=0}^k \check{\lambda}^i |\check{s}(k-i)| \prod_{j=0}^{i-1} \check{c}(k-j), \end{aligned} \quad (4.117)$$

where K''_{max} results from quantization error including δ_{k+1} . From section 4.2, (4.116), and (4.117), the quantized steady-state dynamic range of the internal cell is bounded by

$$\lim_{k \rightarrow \infty} |\check{r}_{1j}(k)| \leq K''_{max} \frac{|\check{x}_{max}|}{\sqrt{1 - \check{\lambda}^2}} = K''_{max} \check{\mathfrak{R}}. \quad (4.118)$$

The output of the m^{th} row is bounded, under the quantization effect, by

$$\lim_{k \rightarrow \infty} |\check{r}_{1j}(k)| \leq K''_{max} (2\check{\lambda})^{m-1} \check{\mathfrak{R}} \quad (4.119)$$

for the highly fluctuated input and

$$\lim_{k \rightarrow \infty} |\check{r}_{1j}(k)| \leq K''_{max} (\check{\lambda})^{m-1} \check{\mathfrak{R}} \quad (4.120)$$

for smooth input.

From these results, the quantized asymptotic value of the residual can be obtained as

$$\lim_{k \rightarrow \infty} |\check{e}(k)| \leq K''_{max} (2\check{\lambda})^{p-1} \check{\mathfrak{R}}. \quad (4.121)$$

Thus, if $\lambda \leq 1$ and the input data are bounded, the QRD LS systolic array constitute a BIBO stable system under the quantization effect.

4.4 Finite-length Effect of Fault-tolerant Capability

In this section, we discuss the finite-length effects of fault-tolerant capability. The first problem is that of missing error detection which results from the cumulative multiplications of the cosine value with a small error. Under a finite-precision implementation, this may result in a failure of error detection. The minimum word-length to circumvent this problem is then derived. The second problem is called the false alarm. With the quantization effects, the system without fault may produce quantization error to cause the false alarm. A threshold device is then introduced to tackle this problem.

4.4.1 Missing Error Detection

By *missing error detection* we mean that a small error generated by a faulty processing cell is not detected due to the finite-precision computation. Assume a fault occurs in an internal cell $PE_{ij}, i \neq j$, at a faulty moment. The output of this faulty cell is thus erroneous and can be described by $x_{out}^e = x_{out} + \delta$, where x_{out} is the fault-free output and δ is the error generated by the fault. The error propagation path can be described by

$$PE_{ij} \rightarrow PE_{(i+1)j} \rightarrow \cdots \rightarrow PE_{jj},$$

and then $PE_{kl}, k \geq j, l \geq j$ are all contaminated. From the operations executed by the internal cell, the error is modified to $c_{i+1}\delta$ by $PE_{(i+1)j}$ and the cumulative

modifications of the error before reaching the boundary cell, PE_{jj} , is

$$\eta = \delta \prod_{k=i+1}^{j-1} c_k, \quad (4.122)$$

where c_i is the cosine parameter generated by the boundary cell PE_{ii} . Let c'_j and s'_j denote the erroneous c_j and s_j respectively. The c'_j and s'_j are then given by

$$c'_j = \frac{\lambda r}{\sqrt{\lambda^2 r^2 + (x_{in} + \eta)^2}}, \quad s'_j = \frac{x_{in} + \eta}{\sqrt{\lambda^2 r^2 + (x_{in} + \eta)^2}}. \quad (4.123)$$

In this case, s'_j is no longer proportional to x_{in} , $\underline{a}(j)$ will not be zeroed out by the j^{th} cell of the EDA. The size of the error generated by this cell is

$$\eta_j = c'_j x_{in} - s'_j \lambda r = -\frac{\lambda r \eta}{\sqrt{r'^2 + 2\eta x_{in} + \eta^2}} = -c'_j \eta, \quad (4.124)$$

where $r' = \sqrt{\lambda^2 r^2 + x_{in}^2}$ is the new updated uncontaminated value of the content of PE_{jj} . When η_j propagates down to the output of the EDA, η_j is influenced by the contaminated cosines c' of each following row. The error output at e_0 due to an error δ generated at PE_{ij} is then given by

$$\begin{aligned} e_0^\delta(i, j) &= -\gamma \prod_{m=j+1}^p c'_m \eta_j = -\gamma \prod_{m=j}^p c'_m \eta \\ &= -\gamma \prod_{k=i+1}^{j-1} c_k \cdot \prod_{m=j}^p c'_m \delta, \end{aligned} \quad (4.125)$$

where $\gamma = \prod_{l=1}^{j-1} c_l \prod_{k=j}^p c'_k [1]$. It becomes

$$e_0^\delta(i, j) = -\prod_{l=1}^i c_l \prod_{k=i+1}^{j-1} c_k^2 \prod_{m=j}^p c_m'^2 \delta. \quad (4.126)$$

Next, assume a fault occurs in a boundary cell, PE_{jj} , $1 \leq j \leq p$, at the faulty moment. Both erroneous c'_j and s'_j produced by PE_{jj} can be written by

$$c'_j = \frac{\lambda r + \delta_c}{r'_c}, \quad s'_j = \frac{x_{in} + \delta_s}{r'_c}, \quad (4.127)$$

where δ_c and δ_s represent errors in the numerators while r'_c represents the erroneous content of the denominators of c_j and s_j . The error produced by the j^{th} cell of the EDA is then given by

$$\eta_j = c'_j x_{in} - s'_j \lambda r = \frac{x_{in} \delta_c - \lambda r \delta_s}{r'_c}, \quad (4.128)$$

and the output error at e_0 due to a faulty boundary cell is given by

$$\begin{aligned} e_0^\delta(j, j) &= \gamma \prod_{m=j+1}^p c'_m \cdot \frac{x_{in} \delta_c - \lambda r \delta_s}{r'_c} \\ &= \prod_{l=1}^j c_l \cdot \prod_{m=j+1}^p c_m'^2 \cdot \eta_j. \end{aligned} \quad (4.129)$$

From (4.126) and (4.129), we can see that $e_0^\delta \neq 0$, under unlimited precision condition, if there is a fault occurring in the system, except when $u_{in} \delta_c = \lambda r \delta_s$ in (4.128). However, this is unlikely to happen. From *Lemma 3.3*, for $0 < c_i \leq 1$, the error may not be detected after multiple multiplications of c_i in (4.126) and (4.129) under finite-precision implementation. It is obvious there is no such problem when δ is large. Since r in (4.123) tends to be a large number asymptotically, it is reasonable to assume the error size δ generated by a fault is much smaller than r when δ is small. Under this circumstance, from (4.123), we have $c'_j \cong c_j$. In the quasi steady-state, the asymptotic behavior of erroneous cosine is $c'_j \cong c_j = \lambda$. From (4.126) and (4.129), the error output e_0^δ due to an error size δ is then approximated by

$$e_0^\delta(i, j) \cong -\lambda^{2p-i} \delta \quad (4.130)$$

for a faulty internal cell and

$$e_0^\delta(j, j) \cong \lambda^{2p-j} \eta_j \quad (4.131)$$

for a faulty boundary cell. Denote B_Δ be the word-length of each memory of fixed point arithmetics. That is, each memory size is of B_Δ bits and let $\Delta = \min(\delta, \eta_j)$.

To insure the detection of error of size Δ , we need

$$\lambda^{2p-i}\Delta \geq \lambda^{2p} \geq 2^{-B_\Delta}. \quad (4.132)$$

Therefore, the memory size should be at least

$$B_\Delta \geq \lceil -2p \log_2 \lambda - \log_2 \Delta \rceil \quad (4.133)$$

such that the small error size can be detected. The second term of the right-hand size is obvious since the error size Δ must be detected; the first term is to account for the effects that the error propagates through the array.

4.4.2 False Alarm

Due to the finite-precision implementation, the residual output of the EDA will not be an actual zero if there is no fault in the system. We call this effect a false alarm. Here, we are going to model and quantitatively describe the false alarm effect and introduce a threshold device to overcome this problem.

Cancellation Principle

Suppose now we have a QRD RLS array of order $p = 3$. Denote the first and second rows of data input as $(x_1, x_2, x_3, x_1 + x_2 + x_3)$ and $(x'_1, x'_2, x'_3, x'_1 + x'_2 + x'_3)$ respectively, where the checksums $x_1 + x_2 + x_3$ and $x'_1 + x'_2 + x'_3$ are inputs to the EDA. Fig. 4.23 shows the first two rows of the array. After both data pass through the array, according to the operations of the processing cells, the contents of the cells of the first row are

$$\begin{aligned} r_{11} &= \sqrt{x_1^2 + x_1'^2}, \\ r_{12} &= sx'_2 + cx_2, \end{aligned}$$

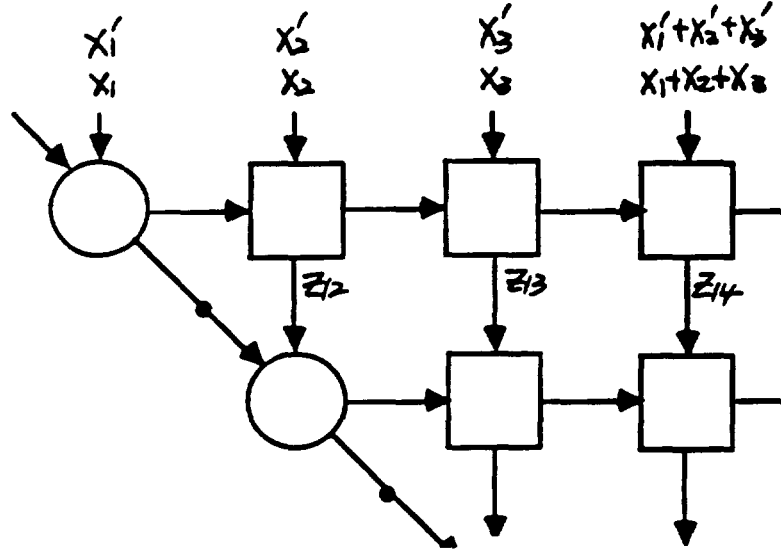


Figure 4.23: The first two rows of the array

$$\begin{aligned} r_{13} &= sx'_3 + cx_3, \\ r_{14} &= s(x'_1 + x'_2 + x'_3) + c(x_1 + x_2 + x_3), \end{aligned} \quad (4.134)$$

where $c = x_1/r_{11}$ and $s = x'_1/r_{11}$ are the rotation parameters generated by the boundary cell and r_{ij} is the content of PE_{ij} . The output of the internal cells are

$$\begin{aligned} z_{12} &= cx'_2 - sx_2, \\ z_{13} &= cx'_3 - sx_3, \\ z_{14} &= c(x'_1 + x'_2 + x'_3) - s(x_1 + x_2 + x_3). \end{aligned} \quad (4.135)$$

Since $sx'_1 + cx_1 = \sqrt{x_1^2 + x'^2_1}$ and $cx'_1 - sx_1 = 0$, we have $r_{14} = r_{11} + r_{12} + r_{13}$ and $z_{14} = z_{12} + z_{13}$. That is, both the contents and the outputs of the first row still meet the checksum. The output of the first cell of EDA, z_{14} , can be rewritten as

$$z_{14} = c(x'_2 + x'_3) - s(x_2 + x_3). \quad (4.136)$$

We can see that the data from the first column got cancelled out by the first cell of the EDA. Since the outputs meet the checksum, with the same principle, the data

from the second column will get cancelled out by the 2nd cell of EDA. Thus, this observation can be generalized and stated as bellowed:

Cancellation Principle: With the checksum encoding data inputted to EDA, the data from the i^{th} column got cancelled out by the i^{th} cell of the EDA. \square

For a finite-precision implementation, due to the roundoff error, the data from the i^{th} column will not be totally cancelled out by the i^{th} cell of the EDA. This effect results in the false alarm problem.

Floating Point Arithmetics

A floating point number f can be represented by [30]

$$f = \pm .d_1 d_2 \cdots d_t \times \beta^e, \quad 0 \leq d_i < \beta, d_1 \neq 0, L \leq e \leq U, \quad (4.137)$$

where β is the base, t is the precision, and $[L, U]$ is the exponent range. The floating point operator fl can be shown to satisfy [30]

$$\begin{aligned} \hat{x} = fl(x) &= x(1 + \epsilon), \\ fl(a \text{ op } b) &= (a \text{ op } b)(1 + \epsilon) \quad |\epsilon| \leq u, \end{aligned} \quad (4.138)$$

where u is the unit roundoff defined by

$$u = \frac{1}{2}\beta^{1-t}, \quad \text{for rounded arithmetics,}$$

and op denote any of the four arithmetic operations $+$, $-$, \times , \div .

Roundoff Analysis

For a QRD RLS array of order p with floating point arithmetics, denote the first row of the input vector as $(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_p, \sum_{i=1}^p \hat{x}_i + \epsilon_p)$, where $\hat{x}_i = fl(x_i)$, $\epsilon_p = \epsilon(\sum_{i=1}^p \hat{x}_i)$,

and $|\epsilon| < \mathbf{u}$ is a constant², and the second row of input vector as $(\hat{x}'_1, \hat{x}'_2, \dots, \hat{x}'_p, \sum_{i=1}^p \hat{x}'_i + \epsilon_p)$. The content of the first boundary cell is given by

$$\hat{r}_{11} = fl(\sqrt{\hat{x}_1^2 + \hat{x}'_1^2}) = \sqrt{\hat{x}_1^2 + \hat{x}'_1^2}(1 + \epsilon), \quad (4.139)$$

and the rotation parameters are $\hat{c} = fl(\hat{x}_1/\hat{r}_{11})$ and $\hat{s} = fl(\hat{x}'_1/\hat{r}_{11})$. The contents of the internal cells then can be obtained as

$$\begin{aligned} \hat{r}_{ij} &= fl(fl(\hat{s}\hat{x}'_j) + fl(\hat{c}\hat{x}_j)) \\ &= [\hat{s}\hat{x}'_j(1 + \epsilon) + \hat{c}\hat{x}_j(1 + \epsilon)](1 + \epsilon) \\ &\approx (1 + 2\epsilon)(\hat{s}\hat{x}'_j + \hat{c}\hat{x}_j), \quad 1 < j \leq p, \end{aligned} \quad (4.140)$$

and

$$\begin{aligned} \hat{r}_{1,p+1} &= fl(fl(\hat{s}(\sum_{i=1}^p \hat{x}'_i + \epsilon_p)) + fl(\hat{c}(\sum_{i=1}^p \hat{x}_i + \epsilon_p))) \\ &\approx (\hat{s} \sum_{i=1}^p \hat{x}'_i + \hat{c} \sum_{i=1}^p \hat{x}_i) + 6\epsilon_p. \end{aligned} \quad (4.141)$$

The mismatch τ_1 resulting from the finite precision computations of the first row is

$$\tau_1 = 6\epsilon_p - (\epsilon\sqrt{\hat{x}_1^2 + \hat{x}'_1^2} + 2\epsilon \sum_{i=2}^p (\hat{s}\hat{x}'_i + \hat{c}\hat{x}_i)) \quad (4.142)$$

and it is bounded by

$$\begin{aligned} |\tau_1| &\leq 6p|\epsilon x_{max}| + |2\epsilon x_{max}| + 4(p-1)|\epsilon x_{max}| \\ &= (10p-2)|\epsilon x_{max}| \leq 10p|\epsilon x_{max}|. \end{aligned} \quad (4.143)$$

For the second row, using the same approach, the mismatch is bounded by $10(p-1)|\epsilon x_{max}|$. The total mismatch from the whole array for a given input vector is given

²To simplify the notation, we do not give indeices to different ϵ 's.

by

$$|\tau| \leq \sum_{i=0}^{p-1} 10(p-i)|\epsilon x_{max}| = 5p(p+1)|\epsilon x_{max}|. \quad (4.144)$$

Let n denote the number of input data rows. As pointed out in [109], for the Givens rotation method, the roundoff error is proportional to $O(n^{1.5})$. The possible mismatch becomes

$$|\tau(n)| \leq 5O(n^{1.5})p(p+1)|\epsilon x_{max}|. \quad (4.145)$$

This bound can be interpreted as: for each row of input, each processing cell contributes about $|\epsilon x_{max}|$ amount of roundoff error. Since there are about $p(p+1)$ processing cells, the total possible roundoff error is then $p(p+1)|\epsilon x_{max}|$.

In order to prevent false alarm, the threshold th has to be set at least at $|\tau(n)|$. Suppose $\beta = 2, t = 16$, then $\mathbf{u} = 2^{-16}$. Given an scaled input data such that $|x_{max}| = 1$ and $n = 10^4$, the threshold of a QRD RLS array of order $p = 100$ is

$$th \geq |\tau(n)|_{max} \simeq 5 \cdot 10^6 \cdot 10^4 \cdot 2^{-16} = 10^{-4}. \quad (4.146)$$

4.4.3 Overall Memory Size Consideration

To prevent missing error detection, we want to set the detectable error size $\Delta = \min(\delta, \eta_j)$ as small as possible. While to prevent the false alarm, we also want to choose a sufficiently high threshold. Both situations cannot be satisfied simultaneously since both goals are conflicting.

To detect the error size Δ , from (4.130), (4.131), and (4.132), we set the threshold $th \leq \lambda^{2p}\Delta$. That is, the minimal detectable error size is $\lambda^{-2p} \cdot th$. From (4.133),

$$B_{\Delta} \leq \lceil -\log_2 th \rceil. \quad (4.147)$$

A criterion to choose B_Δ is then given by

$$B_\Delta = \min(\lceil -2p \log_2 \lambda - \log_2 \Delta \rceil, \lceil -\log_2 th \rceil). \quad (4.148)$$

To prevent overflow, from (4.104), set

$$B_p = \lceil (p-1)(1 + \log_2 \lambda) + \log_2 \mathfrak{R} \rceil. \quad (4.149)$$

For a QRD RLS systolic array to detect an error size of Δ without false alarm and overflow problems, the memory size B is required to be at least

$$B = \max(B_p, B_\Delta). \quad (4.150)$$

Chapter 5

Order Degraded Performance and Residual Estimations

When the faulty row is found in the fault-tolerant QRD RLS systolic array, it enters into an order degraded operation. In this chapter, the performance degradation of the reduced-order QRD LS is studied. The optimal residual estimation under faulty situation is also considered in this chapter.

5.1 Order Degraded Performance

Consider a order p LS problem with a $n \times p$ complex-valued data matrix $A_p(n)$ denoted by

$$A_p(n) = [\underline{u}(1), \underline{u}(2), \dots, \underline{u}(n)]^T = [\underline{a}(1), \underline{a}(2), \dots, \underline{a}(p)] = [A_{p-1}(n) : \underline{a}(p)], \quad (5.151)$$

a $n \times 1$ desired response vector

$$\underline{y}(n) = [d(1), d(2), \dots, d(n)]^T,$$

a $p \times 1$ weight vector

$$\underline{w}_p(n) = [w_1^p(n), w_2^p(n), \dots, w_p^p(n)]^T = [\underline{w}_{p-1|p}(n) : w_p^p(n)]^T, \quad (5.152)$$

and a $n \times 1$ residual vector

$$\underline{\epsilon}_p(n) = [e_p(1), e_p(2), \dots, e_p(n)]^T = A_p(n)\underline{w}_p(n) - \underline{y}(n). \quad (5.153)$$

Let the index of performance be defined by the weighted l_2 norm of

$$\xi_p(n) = \|\underline{\epsilon}_p(n)\|_{\Lambda}^2 = \|\Lambda \underline{\epsilon}(n)_p\|^2 = \underline{\epsilon}_p^H(n) \bar{\Lambda}(n) \underline{\epsilon}_p(n), \quad (5.154)$$

where

$$\Lambda(n) = \text{diag}[\lambda^{(n-1)}, \lambda^{(n-2)}, \dots, \lambda, 1]$$

with a real-valued forgetting factor $0 < \lambda \leq 1$ and $\bar{\Lambda} = \Lambda^2$. Then the least-squares solution, satisfies

$$\xi_{p_{\min}}(n) = \min_{\underline{w}} \|\underline{\epsilon}_p(n)\|_{\Lambda}^2 = \|A_p(n)\hat{\underline{w}}_p(n) - \underline{y}(n)\|_{\Lambda}^2. \quad (5.155)$$

The optimal weight vector can be obtained by solving the normal equation

$$\phi_p(n)\hat{\underline{w}}_p(n) = \psi_p(n), \quad (5.156)$$

where

$$\begin{aligned} \phi_p(n) &= A_p^H(n) \bar{\Lambda} A_p(n) = \begin{bmatrix} A_{p-1}^H(n) \bar{\Lambda} A_{p-1}(n) & : & A_{p-1}^H(n) \bar{\Lambda} \underline{a}_p(n) \\ \text{---} & & \text{---} \\ \underline{a}_p^H(n) \bar{\Lambda} A_{p-1}(n) & : & \underline{a}_p^H(n) \bar{\Lambda} \underline{a}_p(n) \end{bmatrix} \\ &= \begin{bmatrix} \phi_{p-1}(n) & : & \underline{\vartheta}(n) \\ \text{---} & & \text{---} \\ \underline{\vartheta}^H(n) & : & \eta(n) \end{bmatrix} \end{aligned} \quad (5.157)$$

and

$$\psi_p(n) = A_p^H(n) \bar{\Lambda} \underline{y}(n). \quad (5.158)$$

It can be easily shown that

$$\xi_{p_{\min}}(n) = \|\underline{y}(n)\|_{\Lambda}^2 - \psi_p^H(n) \hat{\underline{w}}_p(n). \quad (5.159)$$

For a order $p-1$ LS problem, then as before, we want to minimize the weighted l_2 norm of

$$\underline{\xi}_{p-1}(n) = [e_{p-1}(1), e_{p-1}(2), \dots, e_{p-1}(n)]^T = A_{p-1}(n) \underline{w}_{p-1}(n) - \underline{y}(n) \quad (5.160)$$

with weihgt vector

$$\underline{w}_{p-1}(n) = [w_1^{p-1}(n), w_2^{p-1}(n), \dots, w_{p-1}^{p-1}(n)]^T.$$

Obviously, the optimal weight vector of order p and $p-1$ can be obtained as $\hat{\underline{w}}_p(n) = \phi_p^{-1}(n) \psi_p(n)$ and $\hat{\underline{w}}_{p-1}(n) = \phi_{p-1}^{-1}(n) \psi_{p-1}(n)$ respectively. The *difference vector* of the optimal residual vectors of different order is defined by

$$\underline{\Delta}(n) = \hat{\underline{\xi}}_p(n) - \hat{\underline{\xi}}_{p-1}(n) = A_{p-1}(n) [\hat{\underline{w}}_{p-1|p}(n) - \hat{\underline{w}}_{p-1}(n)] + \hat{w}_p^p(n) \underline{a}(p), \quad (5.161)$$

and the weighted l_2 norm of $\underline{\Delta}(n)$ is defined to be the order degraded performance, $\Gamma(n)$, given by

$$\begin{aligned} \Gamma(n) &= \|\underline{\Delta}(n)\|_{\Lambda}^2 \\ &= \Delta \hat{\underline{w}}^H(n) \phi_{p-1}(n) \Delta \hat{\underline{w}}(n) + \|\hat{w}_p^p(n)\|^2 \cdot \|\underline{a}(p)\|_{\Lambda}^2 \\ &\quad + 2 \cdot \text{Re}[\hat{w}_p^p(n) \Delta \hat{\underline{w}}^H(n) A_{p-1}^H \bar{\Lambda} \underline{a}(p)], \end{aligned} \quad (5.162)$$

where $\Delta \hat{\underline{w}}^H(n) = \hat{\underline{w}}_{p-1|p}^H(n) - \hat{\underline{w}}_{p-1}^H(n)$.

To relate $\hat{\underline{w}}_{p-1|p}(n)$ with $\hat{\underline{w}}_{p-1}(n)$, we define a LS problem of order $p-1$ with A_{p-1} as the data matrix and $\underline{a}(p)$ as the desired response. That is, we would like to minimize the weighted l_2 norm of the residual vector

$$\underline{\xi}_{a,p-1}(n) = A_{p-1}(n)\underline{w}_{a,p-1}(n) - \underline{a}(p) = [e_{a,p-1}(1), e_{a,p-1}(2), \dots, e_{a,p-1}(n)]^T. \quad (5.163)$$

From (5.153) and (5.156), the optimal weight vector can be obtained by solving

$$\phi_{p-1}(n)\hat{\underline{w}}_{a,p-1}(n) = \underline{\theta}(n), \quad (5.164)$$

where $\underline{\theta}(n)$ is defined in (5.157). Let the optimal index of performance of this LS problem be $\xi_{a,p-1}(n) = \|\underline{\xi}_{a,p-1}(n)\|_\Lambda^2$, then ϕ_p^{-1} can be represented by

$$\phi_p^{-1}(n) = \begin{bmatrix} \phi_{p-1}^{-1}(n) & \vdots & \underline{0}_{p-1} \\ - & - & - \\ \underline{0}_{p-1}^T & & 0 \end{bmatrix} + \frac{1}{\xi_{a,p-1}(n)} \begin{bmatrix} -\hat{\underline{w}}_{a,p-1}(n) \\ 1 \end{bmatrix} [-\hat{\underline{w}}_{a,p-1}^H(n) \quad 1]. \quad (5.165)$$

Then $\hat{\underline{w}}_p(n)$ can be represented by expressions of order $p-1$ as

$$\begin{aligned} \hat{\underline{w}}_p(n) &= \phi_p^{-1}(n)\psi_p(n) = \phi_p^{-1}(n) \begin{bmatrix} A_{p-1}^H(n) \\ - & - & - \\ \underline{a}^H(p) \end{bmatrix} \bar{\Lambda} \underline{y}(n) \\ &= \begin{bmatrix} \phi_{p-1}^{-1}(n)A_{p-1}^H(n) + \frac{\hat{\underline{w}}_{a,p-1}(n)\underline{\xi}_{a,p-1}^H(n)}{\xi_{a,p-1}(n)} \\ - & - & - & - & - \\ \frac{-\hat{\underline{\xi}}_{a,p-1}^H(n)}{\xi_{a,p-1}(n)} \end{bmatrix} \bar{\Lambda} \underline{y}(n), \end{aligned} \quad (5.166)$$

and therefore,

$$\hat{\underline{w}}_{p-1|p}(n) = \hat{\underline{w}}_{p-1}(n) + \frac{\mathcal{I}(n)}{\xi_{a,p-1}(n)} \hat{\underline{w}}_{a,p-1}(n), \quad (5.167)$$

$$\hat{w}_p^p(n) = -\frac{\mathcal{I}(n)}{\xi_{a,p-1}(n)}, \quad (5.168)$$

where $\mathcal{I}(n) = \langle \hat{\underline{x}}_{a,p-1}(n), \underline{y}(n) \rangle_{\Lambda}$ and $\langle \underline{x}, \underline{y} \rangle_{\Lambda} = \langle \Lambda \underline{x}, \Lambda \underline{y} \rangle$ is the weighted inner product. Thus,

$$\Delta \hat{\underline{w}}(n) = \frac{\mathcal{I}(n)}{\xi_{a,p-1}(n)} \hat{\underline{w}}_{a,p-1}(n). \quad (5.169)$$

Now, we may proceed to calculate $\Gamma(n)$ in (5.162). The first term of (5.162) becomes

$$\begin{aligned} & \Delta \hat{\underline{w}}^H(n) \phi_{p-1}(n) \Delta \hat{\underline{w}}(n) \\ &= \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \hat{\underline{w}}_{a,p-1}(n) A_{p-1}^H(n) \bar{\Lambda} A_{p-1}(n) \hat{\underline{w}}_{a,p-1}(n) \\ &= \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} (\hat{\underline{x}}_{a,p-1}(n) + \underline{a}(p))^H \bar{\Lambda} (\hat{\underline{x}}_{a,p-1}(n) + \underline{a}(p)) \\ &= \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} (\xi_{a,p-1}(n) + \eta(n) + 2 \cdot \text{Re}(\langle \hat{\underline{x}}_{a,p-1}(n), \underline{a}(p) \rangle_{\Lambda})). \end{aligned} \quad (5.170)$$

The second term of (5.162) becomes

$$\|\hat{\underline{w}}_p^p(n)\|^2 \cdot \|\underline{a}(p)\|_{\Lambda}^2 = \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \cdot \eta(n). \quad (5.171)$$

The third term of (5.162) is

$$\begin{aligned} & 2 \cdot \text{Re}[\hat{\underline{w}}_p^p(n) \Delta \hat{\underline{w}}^H(n) A_{p-1}^H \bar{\Lambda} \underline{a}(p)] \\ &= 2 \cdot \text{Re}\left[-\frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \hat{\underline{w}}_{a,p-1}^H(n) A_{p-1}^H(n) \bar{\Lambda} \underline{a}(p)\right] \\ &= -2 \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \cdot \text{Re}[(\hat{\underline{x}}_{a,p-1}(n) + \underline{a}(p))^H \bar{\Lambda} \underline{a}(p)] \\ &= -2 \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \eta(n) - 2 \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \cdot \text{Re}(\langle \hat{\underline{x}}_{a,p-1}(n), \underline{a}(p) \rangle_{\Lambda}). \end{aligned} \quad (5.172)$$

Combining (5.170), (5.171), and (5.172) together, we have

$$\Gamma(n) = \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}(n)} = \frac{\|\langle \hat{\underline{x}}_{a,p-1}(n), \underline{y}(n) \rangle_{\Lambda}\|^2}{\|\hat{\underline{x}}_{a,p-1}(n)\|_{\Lambda}^2}. \quad (5.173)$$

Denote the last row of input data matrix $\underline{u}(n) = [\underline{u}_{p-1}(n), u_p(n)]$, the difference of the optimal residual at time n then can be obtained as

$$\|\hat{e}_p(n) - \hat{e}_{p-1}(n)\|^2 = \|\underline{u}^T(n) \hat{\underline{w}}_p(n) - \underline{u}_{p-1}^T(n) \hat{\underline{w}}_{p-1}(n)\|^2$$

$$\begin{aligned}
&= \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \|\underline{u}_{p-1}^T(n) \hat{\underline{w}}_{a,p-1}(n) - u_p(n)\|^2 \\
&= \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}^2(n)} \|\hat{\underline{e}}_{a,p-1}(n)\|^2.
\end{aligned} \tag{5.174}$$

5.1.1 Geometric Interpretation

From (5.173), we can see the order degraded performance is indeed the energy of the projection of the desired response $\underline{y}(n)$ onto the subspace spanned by the optimal residual $\hat{\underline{e}}_{a,p-1}(n)$. As the vector $\underline{y}(n)$ becomes more orthogonal to $\hat{\underline{e}}_{a,p-1}(n)$, the order degraded performance is also reduced. Denote the column space of $A_{p-1}(n)$ by $\{A_{p-1}(n)\}$. Then the projection operator $P_{A_{p-1}}$ projects vectors onto space $\{A_{p-1}(n)\}$ and the orthogonal projection operator $P_{A_{p-1}}^\perp = I - P_{A_{p-1}}$ projects vectors onto the space $\{A_{p-1}^\perp(n)\}$ which is orthogonal to space $\{A_{p-1}(n)\}$. The entire space S spanned by $\underline{y}(n)$ can be represented by

$$S = \{A_{p-1}(n)\} \cup \{P_{A_{p-1}}^\perp \underline{a}(p)\} \cup \{A_p^\perp(n)\}, \tag{5.175}$$

and all of these subspaces are orthogonal to each other. It is obvious that $\hat{\underline{e}}_{a,p-1}(n) = P_{A_{p-1}}^\perp \underline{a}(p)$. By projecting the desired response $\underline{y}(n)$ to these subspaces, we obtain

$$\begin{aligned}
\underline{y}(n) &= P_{A_{p-1}} \underline{y}(n) + P_{\hat{\underline{e}}_{a,p-1}} \underline{y}(n) + P_{A_p^\perp} \underline{y}(n) \\
&= \hat{\underline{y}}(n) + \langle \hat{\underline{e}}_{a,p-1}(n), \underline{y}(n) \rangle_\Lambda \frac{\hat{\underline{e}}_{a,p-1}(n)}{\|\hat{\underline{e}}_{a,p-1}(n)\|_\Lambda^2} + \hat{\underline{e}}_p(n),
\end{aligned} \tag{5.176}$$

and all of these vectors are also orthogonal to each other. If we drop the vector $\underline{a}(p)$, then the one-dimensional subspace of $\{P_{A_{p-1}}^\perp \underline{a}(p)\}$ cannot be used to represent $\underline{y}(n)$. Therefore, the components of $\underline{y}(n)$ in this subspace is lost and this introduces an error vector $P_{\hat{\underline{e}}_{a,p-1}} \underline{y}(n)$ with energy

$$\|P_{\hat{\underline{e}}_{a,p-1}} \underline{y}(n)\|^2 = \frac{\|\mathcal{I}(n)\|^2}{\xi_{a,p-1}(n)} = \Gamma(n). \tag{5.177}$$

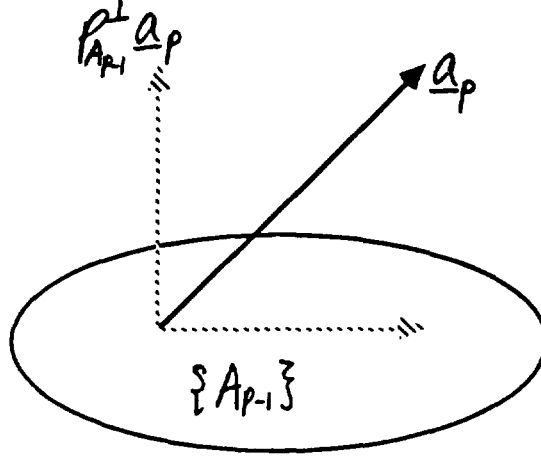


Figure 5.24: Geometric illustration

The energy of the last component of this error vector is given by (5.174). Fig 5.24 illustrates the geometric interpretation discussed above.

5.2 Residual Estimation in Faulty Situation

From Section 4.4, we know that if there is a fault occurring in the system, the error output at e_0 due to an error δ generated at an internal cell PE_{ij} is given by

$$e_0^\delta(i, j) = -\gamma \prod_{m=j}^p c'_m \eta, \quad (5.178)$$

where $\eta == \delta \prod_{k=i+1}^{j-1} c_k$, and the error due to a faulty boundary cell is given by

$$e_0^\delta(j, j) = \prod_{l=1}^j c_l \cdot \prod_{m=j+1}^p c_m'^2 \cdot \eta_j, \quad (5.179)$$

where $\eta_j = (x_{in}\delta_c - \lambda r\delta_s)/r'_c$ with δ_c and δ_s representing errors in the numerators while r'_c represents the erroneous content of the denominators of c_j and s_j . Whenever $e_0^\delta(n) \neq 0$, we detect there is a fault occurring in the system. The optimal residual

$\hat{e}(n)$ is thus erroneous and is denoted as $\hat{e}^\delta(n)$. In this section, we devise a method to estimate the true optimal residual $\hat{e}(n)$ given the contaminated residual $\hat{e}^\delta(n)$.

5.2.1 Faulty Internal Cell

Once the faulty cell PE_{ij} is identified, then η in (5.178) can be estimated by

$$\eta = -\frac{e_0^\delta(n)}{\gamma \prod_{m=j}^p c'_m}, \quad (5.180)$$

where c'_m s are obtained at the right hand side of the systolic array. If η is large, then through many non-linear operations in the cells, it is essentially impossible to keep track of the effect of η on $e^\delta(n)$. Thus, for analytical tractability, we assume the error η is small in the sense that $\eta x^p \ll r$. With this assumption, the erroneous cosine can be approximated by

$$\begin{aligned} c'_j &= \frac{\lambda r}{\sqrt{\lambda^2 r^2 + (x + \eta)^2}} \\ &\simeq \frac{\lambda r}{\sqrt{r'^2 + 2\eta x}} \simeq \frac{\lambda r}{r' + \eta x} \\ &= \frac{c_j}{1 + \frac{\eta x}{r'}} \simeq c_j - \frac{\eta x}{r'} = c_j - \eta s_j \end{aligned} \quad (5.181)$$

and the erroneous sine is given by

$$\begin{aligned} s'_j &= \sqrt{1 - c_j'^2} \simeq \sqrt{(1 - c_j^2) + 2\eta s_j c_j} \\ &\simeq s_j + \eta s_j c_j. \end{aligned} \quad (5.182)$$

And since $\eta x^p \ll r$, we also have $\eta s_j \ll 1$ and $\eta s_j c_j \ll c_j < 1$ which yield $c'_j < 1$, $s'_j < 1$, and $c_j'^2 + s_j'^2 \simeq 1$.

The erroneous cosine and sine generated by the boundary cell PE_{jj} are then sent to the right for processing by the internal cells $PE_{j,j+k}$, $1 \leq k \leq p-j$. The output

of these cells are erroneous and are given by

$$\begin{aligned}
x'_{out,j,j+k} &= c'_j x - s'_j \lambda r_{j,j+k} \\
&= (c_j - \eta s_j) x - (s_j + \eta s_j c_j) \lambda r_{j,j+k} \\
&\simeq x_{out,j,j+k} - \eta s_j c_j \lambda r_{j,j+k},
\end{aligned} \tag{5.183}$$

where x is the input data and x_{out} and x'_{out} are the uncontaminated and erroneous output respectively. Denote the error size produced by cell $PE_{i,j}$ as $\mathfrak{S}_{i,j}$. With this notation, $\mathfrak{S}_{j-1,j} = \eta$, and the error size produced by cell $PE_{j,j+k}$ is

$$\mathfrak{S}_{j,j+k} = \mathfrak{S}_{j-1,j} s_j c_j \lambda r_{j,j+k}, \quad 1 \leq k \leq p-j. \tag{5.184}$$

Recall that $O(r) \sim O(x/\sqrt{1-\lambda^2})$ and $s \sim \sqrt{1-\lambda^2}$, we have $O(\mathfrak{S}_{j,j+k}) \sim O(\eta x)$. As discussed above, the contaminated rotation parameters generated by the boundary cell $PE_{j+k,j+k}$, are given by

$$\begin{cases} c'_{j+k} = c_{j+k} - \mathfrak{S}_{j+k-1,j+k} s_{j+k} \\ s'_{j+k} = s_{j+k} + \mathfrak{S}_{j+k-1,j+k} s_{j+k} c_{j+k} \end{cases} \quad 1 \leq k \leq p-j. \tag{5.185}$$

By solving this linear system of equations, we can estimate the uncontaminated sine and cosine as

$$\begin{cases} c_{j+k} \simeq \frac{1}{2}(c'_{j+k} - \mathfrak{S}_{j+k-1,j+k}^{-1} + \sqrt{(c'_{j+k} - \mathfrak{S}_{j+k-1,j+k}^{-1})^2 + 4s'_{j+k}}) \\ s_{j+k} \simeq \sqrt{1 - c_{j+k}^2} \end{cases} \quad 0 \leq k \leq p-j. \tag{5.186}$$

When $\mathfrak{S}_{j+k-1,j+k}^{-1} \ll 1$, we can further simply (5.186) as

$$c_{j+k} = c'_{j+k} + s'_{j+k}. \tag{5.187}$$

To obtain $\mathfrak{S}_{j+k,j+l}$, $1 \leq k \leq p-j$, $k \leq l \leq p-j$, from the operations of the internal cell

$$x'_{out,j+k,j+l}$$

$$\begin{aligned}
&= (c_{j+k} - \mathfrak{S}_{j+k-1,j+k}s_{j+k})(x + \mathfrak{S}_{j+k-1,j+l}) - (s_{j+k} + \mathfrak{S}_{j+k-1,j+k}s_{j+k}c_{j+k}) \cdot \lambda r_{j+k,j+l} \\
&= x_{out,j+k,j+l} - (s_{j+k}\mathfrak{S}_{j+k-1,j+k} - c_{j+k})\mathfrak{S}_{j+k-1,j+l} - \mathfrak{S}_{j+k-1,j+k}s_{j+k}c_{j+k}\lambda r_{j+k,j+l} \quad (5.188)
\end{aligned}$$

A recursive equation is then obtained to compute $\mathfrak{S}_{j+k,j+l}$ from $\mathfrak{S}_{j+k-1,j+l}$ and $\mathfrak{S}_{j+k-1,j+k}$ as

$$\begin{aligned}
\mathfrak{S}_{j+k,j+l} &= \rho_{j+k}\mathfrak{S}_{j+k-1,j+l} + \mathfrak{S}_{j+k-1,j+k}s_{j+k}c_{j+k}\lambda r_{j+k,j+l}, \\
1 \leq k \leq p-j, k < l \leq p-j,
\end{aligned} \tag{5.189}$$

where $\rho_{j+k} = s_{j+k}\mathfrak{S}_{j+k-1,j+k} - c_{j+k}$. The order of $\mathfrak{S}_{j+k,j+l}$ is that of

$$O(\mathfrak{S}_{j+k-1,j+k}s_{j+k}c_{j+k}\lambda r_{j+k,j+l}).$$

Therefore, $O(\mathfrak{S}_{j+1,j+l}) \sim O(\eta x^2)$. We can also show that $O(\mathfrak{S}_{j+k,j+l}) \sim O(\eta x^k) < O(\eta x^p)$. Thus the error size is consistent with the original assumption of the analyzable situation.

With (5.189), we can recursively compute the error size produced by the cell $PE_{p,p+1}$ and that of $\mathfrak{S}_{p,p+1}$. The erroneous residual can be obtained as

$$e^\delta(n) = e(n) - \gamma \mathfrak{S}_{p,p+1}. \tag{5.190}$$

The optimal uncontaminated residual is then estimated as

$$e(n) = e^\delta(n) + \gamma \mathfrak{S}_{p,p+1}. \tag{5.191}$$

To update the content of $PE_{j+k,j+k}$, we try to keep it as uncontaminated as possible by performing

$$\begin{aligned}
r_{j+k,j+k}(n+1) &= (\lambda^2 r_{j+k,j+k}^2(n) + (x + \mathfrak{S}_{j+k-1,j+k}))^{\frac{1}{2}}, \\
r_{j+k,j+l}(n+1) &= s_{j+k}(x + \mathfrak{S}_{j+k-1,j+l}) - c_{j+k}\lambda r_{j+k,j+l}(n), \\
1 \leq k \leq p-j, k < l \leq p-j.
\end{aligned} \tag{5.192}$$

We summarize the recursive estimation procedure as follows,

1. Estimate error size $\eta = -e_0^\delta(n)/(\gamma \prod_{m=j}^p c'_m)$ from the EDA.
2. Estimate the uncontaminated rotation parameters by using

$$\begin{cases} c_{j+k} \simeq \frac{1}{2}(c'_{j+k} - \mathfrak{S}_{j+k-1,j+k}^{-1} + \sqrt{(c'_{j+k} - \mathfrak{S}_{j+k-1,j+k}^{-1})^2 + 4s'_{j+k}} \\ s_{j+k} \simeq \sqrt{1 - c_{j+k}^2} \end{cases} \quad 0 \leq k \leq p-j.$$

3. Recursive estimation of the error size $\mathfrak{S}_{j+k,j+l}$ by computing

$$\mathfrak{S}_{j+k,j+l} = \rho_{j+k} \mathfrak{S}_{j+k-1,j+l} + \mathfrak{S}_{j+k-1,j+k} s_{j+k} c_{j+k} \lambda r_{j+k,j+l}, \quad 1 \leq k \leq p-j, k < l \leq p-j.$$

4. Estimate the output residual from

$$e(n) = e^\delta(n) + \gamma \mathfrak{S}_{p,p+1}.$$

5. Update the contents of the processing cells by doing

$$\begin{aligned} r_{j+k,j+k}(n+1) &= (\lambda^2 r_{j+k,j+k}^2(n) + (x + \mathfrak{S}_{j+k-1,j+k}))^{\frac{1}{2}} \\ r_{j+k,j+l}(n+1) &= s_{j+k}(x + \mathfrak{S}_{j+k-1,j+l}) - c_{j+k} \lambda r_{j+k,j+l}(n), \\ 1 \leq k \leq p-j, k < l \leq p-j. \end{aligned}$$

5.2.2 Faulty Boundary Cell

For a faulty boundary cell, from (5.179), we can estimate

$$\eta_j = \frac{e_0^\delta(j, j)}{\prod_{l=1}^j c_l \cdot \prod_{m=j+1}^p c_m^2}. \quad (5.193)$$

Unfortunately, there is no way to estimate δ_c and δ_s from η_j . Therefore, we are unable to estimate the optimal residual if the faulty cell is a boundary cell.

Chapter 6

Multi-phase Systolic Algorithms for Spectral Decomposition

In this chapter, we propose two multi-phase systolic algorithms to solve the spectral decomposition problem based on QR algorithm. The spectral decomposition constitutes one of the most computationally intensive needs of modern signal processing applications. While the QR algorithm is well known to be an effective method to solve the eigenvalue problem, there is still no single systolic array architecture that can compute the unitary Q matrix readily and perform the QR algorithm efficiently. Previous methods based on the Jacobi-like approach required global communication or broadcast in computing the eigenvector, and methods using the QR algorithm had communication problems among different architectures. In this chapter, we show that the Q matrix can be computed easily by using multi-phase systolic algorithms and thus the eigenvectors can also be computed without any global communication in the array. Two arrays, a triangular and rectangular, are presented to implement

the multi-phase algorithms. Details on these multi-phase operations of the QR algorithm as well as architectural consequences are discussed in the chapter. Efficient fault-tolerant schemes for these multi-phase operations are also considered.

Recent developments in the parallel processing architectures, especially the systolic architectures, for the spectral decomposition are discussed in section 6.1. In section 6.2, some preliminary matrix operations useful for the multi-phase operations are discussed. In section 6.3, we review the QR algorithm and show the evaluation of the eigenvector from cumulative multiplications of the Q matrices. Then two multi-phase systolic arrays for the QR algorithm and the Hessenberg reduction are presented in section 6.4. Their performances, numerical stabilities, and convergence rates are studied in section 6.5. Finally, some efficient fault-tolerant schemes that can be incorporated with the arrays are discussed in section 6.6.

6.1 Recent Developments

Computing the spectral decomposition of a matrix is an important issue in many modern signal processing and system applications. The feasibility of real-time processing for sophisticated modern signal processing systems, depends crucially on efficient implementation of parallel processing of the algorithms and associated architectures needed to perform these operations [14, 58]. While many variations exist in the literatures for solving these matrix problems, the heart of all these iterative methods are based either on the Jacobi-Hestennes method or the QR algorithm [30, 101, 107]. While there are some fundamental differences between these two approaches, both algorithms have good numerical stability and convergence rate properties and thus are desirable for possible implementation. Since present VLSI technology is capable of

building a multiprocessor system on a chip, many researchers have proposed different parallel processing architectures to solve eigenvalue and singular value decomposition (SVD) problems.

For any complex-valued $m \times n$ matrix A , the classical spectral decomposition [102] of the $n \times n$ Hermetian matrix $A^H A$, is given by

$$A^H A = \sum_{i=1}^n \lambda_i \underline{v}_i \underline{v}_i^H = V \Lambda V^H, \quad (6.194)$$

where $V = [\underline{v}_1, \dots, \underline{v}_n]$ is an $n \times n$ unitary matrix, $\Lambda = \text{diag}[\lambda_1, \dots, \lambda_n]$, and H is the complex conjugate transpose operator. The λ_i 's are the eigenvalues satisfying $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$ and the \underline{v}_i 's are the eigenvectors satisfying $A^H A \underline{v}_i = \lambda_i \underline{v}_i$. The decomposition of $A^H A$ follows from the SVD [30] of A given by

$$A = U S V^H, \quad (6.195)$$

where $U = [\underline{u}_1, \dots, \underline{u}_m]$ is an $m \times m$ matrix with orthogonal column vectors, $S = \text{diag}[s_1, \dots, s_n]$, and V is an $n \times n$ unitary matrix. The s_i 's are the singular values satisfying $s_1 \geq s_2 \geq \dots \geq s_n \geq 0$ and are the positive square roots of λ_i 's such that $\Lambda = S^2$. In this paper, we shall use spectral decomposition in the broad sense of not only including the decompositions of (1) and (2), but also include the eigenvalue decomposition of an arbitrary complex-valued $n \times n$ matrix A given by

$$A X = \Lambda X, \quad (6.196)$$

where X is an $n \times n$ matrix of eigenvectors and $\Lambda = \text{diag}[\lambda_1, \dots, \lambda_n]$ is the matrix of eigenvalues of A .

Luk [72], Brent [12], and Gao and Thomas [27], have used effectively the Jacobi-like method to solve these problems for either a multiprocessor system or systolic

array. The basic problem concerns the diagonalization of a 2×2 matrix by the rotation matrices $J(\theta)$ and $K(\phi)$ in

$$J(\theta)^T \begin{bmatrix} w & x \\ y & z \end{bmatrix} K(\phi) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}. \quad (6.197)$$

A two-stage procedure is then used to find θ and ϕ [72]. To find the SVD of a square matrix A , an appropriate sequence of 2×2 matrices is computed by using the basic Jacobi transformation in

$$T_{ij} : A \leftarrow J_{ij}^T A K_{ij} \quad (6.198)$$

where J_{ij} and K_{ij} are rotations in the (i, j) plane chosen to annihilate the (i, j) and (j, i) elements of A respectively [72]. While the Jacobi-like method, as considered in [72], is currently known as one of the most effective parallel SVD algorithm for full dense matrices, the computations required to obtain the rotational matrices needed in this approach to obtain the singular vectors are not simple and can not be obtained without broadcast [72]. Moreno and Lang [82] also considered some alternatives to the algorithms in [12].

On the other hand, other researchers have used QR algorithm to solve the eigenvalue problems. Heller and Ipsen [33, 43] performed the QR iteration for banded matrix based on orthogonal systolic network and Schreiber [94] combined their network with Gentleman-Kung's QR array to compute the QR algorithm. These methods required the computation of the unitary matrix Q . However, problems exist in the concurrent computation of Q and the pipeline operation of the QR iteration [43]. In [81], Moldovan et al. studied the mapping of a large QR algorithm onto a fixed size array. Torralba and Navarro [103] further purposed a size-independent linear array for QR iteration and Hessenberg reduction. While this approach can provide an efficient

computation of one iteration of the QR iteration, it is not obvious how to pipeline the iteration.

For some system applications, such as matrix rank determination and system identification [52], the efficient computation of singular values is sufficient, while in other applications such as antenna beamformation [79, 97], spectral estimation [50, 93], direct finding [31, 85], etc., the eigenvectors are crucially needed. This makes practical implementation of systolic arrays discussed above difficult for many applications since they either cannot compute the eigenvector or cannot obtain the eigenvector without broadcast. For example, for the MUSIC algorithm [32], once we determine the signal subspace and noise subspace from the eigenvalues, the sample spectrum is then determined by

$$S(\omega) = \frac{1}{s^H(\omega)X_N X_N^H s(\omega)},$$

where X_N is the matrix of eigenvectors which generate the noise subspace and

$$s(\omega) = [1, e^{-j\omega}, \dots, e^{-j\omega(K-1)}],$$

with K is the dimension of matrix X_N . A system which consists of several systolic modules to compute the MUSIC algorithm has been proposed in [90]. However, communication problems among the modules and the difficulty of matching the pipeline rates and timings among different modules may pose difficulties for practical implementation.

Presently, there is no known simple efficient systolic array approach for the generation of eigenvectors. The main reason is that there is no single architecture that is capable of handling all the steps required in the algorithm such that we can pipeline the successive iteration readily. The communication cost among different architectures is high and the interface problem for an efficient data flow is demanding. In this

paper, we propose two multi-phase systolic algorithms to solve the spectral decomposition problem based on the QR algorithm. By multi-phase operations we mean that the processing cells can perform different arithmetical operations in different phase of the computations. Two systolic arrays, one is triangular and the other is rectangular, are designed based on the multi-phase concept. A key feature in our method for the successfully application of the QR algorithm is that the Q matrix of the QR decomposition can be computed explicitly by multiphase operations. With the proper feedback of this Q matrix, the QR algorithm can be computed and pipelined effectively in a single systolic array. From the accumulation of those Q matrices in another array, eigenvectors and singular vectors can be computed without needing global communication inside the array.

6.2 Systolic Array Matrix Processing

In this section, we consider some preliminary matrix and associated systolic array operations needed in the multi-phase systolic algorithms for spectral decompositions.

A. QR Decomposition

A non-degenerate $m \times n$ rectangular matrix A can be factorized into two matrices Q and R such that $A = QR$, where Q is an $m \times m$ unitary matrix and R is an $m \times n$ upper triangular matrix. The matrix Q can be computed using sequences of Givens rotations. An elementary Givens transformation has the form of

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} 0 & \cdots & 0 & r_i & r_{i+1} & \cdots & r_k \\ 0 & \cdots & 0 & x_i & x_{i+1} & \cdots & x_k \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & r'_i & r'_{i+1} & \cdots & r'_k \\ 0 & \cdots & 0 & 0 & x'_{i+1} & \cdots & x'_k \end{bmatrix} \quad (6.199)$$

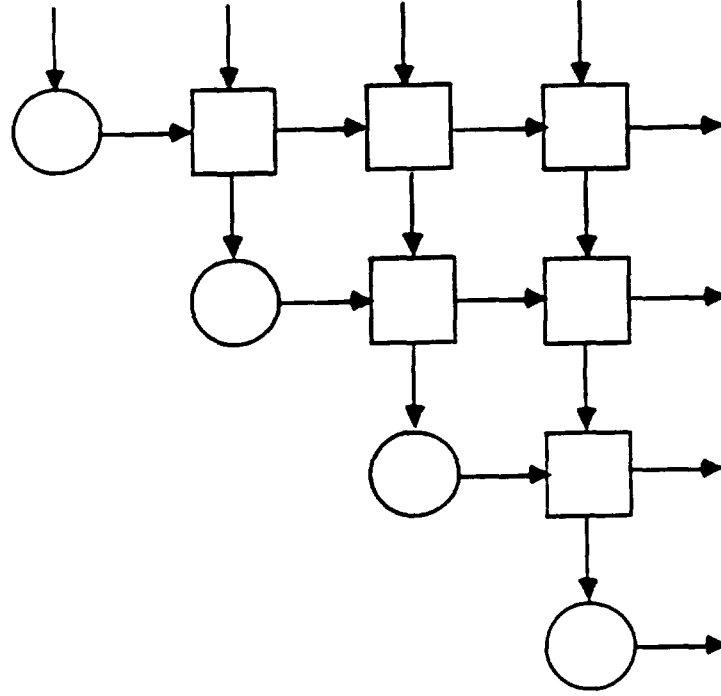


Figure 6.25: Triangular systolic array for QR decomposition.

where

$$c = \frac{r_i}{\sqrt{r_i^2 + x_i^2}}, \quad s = \frac{x_i}{\sqrt{r_i^2 + x_i^2}}.$$

Several different QR arrays have been considered by Gentleman and Kung [26], Heller and Ipsen [31], and Luk [67]. In particular, the computation of the Q matrix without broadcast is difficult for the array considered in [66, pp.266]. On the other hand, [26] has shown that a triangular systolic array can be used to obtain the upper triangular matrix R based on sequences of Givens rotations. This approach also leads to an efficient method for performing recursive least-squares computation [72], and is also useful for finding the singular value of a matrix [25]. This systolic array is shown in Fig. 6.25 and the operations of the cells are described in the first column (*i.e.*, phase 1) of Table 6.5. While the rotation parameters are propagated to the right, the Q matrix will not appear directly at the right as originally suggested by [86]. In order to

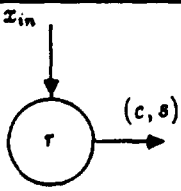
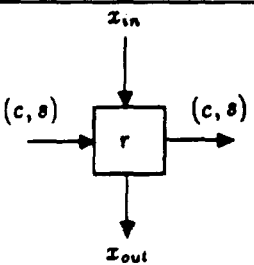
Cell \ Phase	Phase 1	Phase 2	Phase 3
	<p>If $x_{in} = 0$ then $c \leftarrow 1; s \leftarrow 0;$ otherwise $r' = \sqrt{\lambda^2 r^2 + x_{in}^2};$ $c \leftarrow \lambda r / r'; s \leftarrow x_{in} / r';$ $r \leftarrow r';$ end</p>	$s \leftarrow x_{in} / r$	$s \leftarrow x_{in} r$
	$x_{out} \leftarrow c x_{in} - s \lambda r$ $r \leftarrow s x_{in} + c \lambda r$	$x_{out} \leftarrow x_{in} - s r$	$s_{out} \leftarrow s_{in} + x_{in} r$

Table 6.5: Operations of the processing cells for different phases.

demonstrate this point, denote G_{ij} as the Givens rotation matrix of the (i, j) plane, then matrix Q can be obtained as

$$Q^T = \prod_{i=m-1}^1 \prod_{j=i+1}^m G_{ij}, \quad (6.200)$$

where \prod is an ordered matrix product such that $\prod_{i=m-1}^1 C_i = C_{m-1} C_{m-2} \cdots C_1$, while \prod denoted a conventional ordinary product. From Table 6.6, we can see, for a $n \times n$ QR triarray, the first rotation parameter coming out at the right edge occurs at time $n+1$. After that, rotation parameters for different plane rotation come out successively. If assuming that the operation of \prod discussed above can be obtained immediately, then there are m operations of \prod when all of \prod are available. This observation leads to the conclusion that we cannot obtain the Q matrix by cumulatively multiplying the rotation parameters propagated to the right edge. Thus, This is not an effective way to obtain the Q matrix.

Time	n+1	n+2	n+3	n+4	n+5	n+6
First row	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
Second row			(2,3)	(2,4)	(2,5)	(2,6)
Third row					(3,4)	(3,5)

Table 6.6: The timing table for the rotation parameters to reach the right edge of the QR triarray.

B. Computation of $R^{-T}\underline{x}$

In [7], Comon and Robert presented a rectangular systolic array for the computation of $B^{-1}A$, where B and A are square and rectangular matrices respectively. The computation takes two phases. First, the matrix B is fed into the array and B^{-1} is computed. In the second phase, the matrix A is inputted to produce $B^{-1}A$. For the special case where B is an upper triangular matrix denoted by R , instead of a full dense matrix, McWhirter and Shepherd [73] used the property that a triangular array can compute $R^{-T}\underline{x}$ in one phase with the matrix R prestored in the triarray. Since this property is needed in phase 2 of our work, and no derivation was given in [73], we present a brief derivation of this result. Define $r_{ij} = (R)_{ij}$ and $r'_{ij} = (R^{-1})_{ij}$, where $r_{ij} = 0$ and $r'_{ij} = 0$ for $i > j$, then it can be shown that

$$r'_{ij} = \begin{cases} 1/r_{ii}, & i = j, \\ -\sum_{k=i}^{j-1} r'_{ik}r_{kj}/r_{jj}, & i < j \leq n. \end{cases} \quad (6.201)$$

Let $[y_1, \dots, y_n]^T = R^{-T} \underline{x}$, then

$$y_j = \sum_{i=1}^j x_i r'_{ij}, \quad j = 1, \dots, n. \quad (6.202)$$

In particular, y_i can be expressed in terms of r_{ij} as

$$\begin{aligned} y_j &= \frac{1}{r_{jj}} \cdot \left(x_j - \sum_{i=1}^{j-1} x_i \sum_{k=i}^{j-1} r'_{ik} r_{kj} \right) \\ &= \frac{1}{r_{jj}} \cdot \left(x_j - \sum_{k=1}^{j-1} \sum_{i=1}^k x_i r'_{ik} r_{kj} \right). \end{aligned} \quad (6.203)$$

From (6.202), y_j is given by

$$y_j = \underbrace{\frac{1}{r_{jj}}}_{(1)} \underbrace{\left(x_j - \sum_{k=1}^{j-1} y_k r_{kj} \right)}_{(2)} \quad (6.204)$$

Thus, y_i can be computed recursively according to the above equation in the following algorithm: Fig. 6.26 shows the data flow of the input \underline{x} and the output \underline{y} .

Recursive Algorithm for Computing $\underline{y} = R^{-T} \underline{x}$

```

 $y_1 = \frac{1}{r_{11}} \cdot x_1$ 
for  $j = 2$  to  $n$ 
  begin
     $z_j = x_j$ 
    for  $k = 1$  to  $j - 1$ 
       $z_j = z_j - y_k r_{kj}$ 
     $y_j = z_j / r_{jj}$ 
  end

```

The corresponding systolic array to implement the above algorithm is the same as the one shown in Fig. 6.25. The operations of the cells are shown in the second column of Table 6.5. The first part of the (6.204), (*i.e.*, the division), is performed by the boundary cell while the second part of (6.204) is cumulated by the internal cells. With R pre-stored in the triarray, Fig. 6.26 shows the data flow of the input \underline{x} and the output \underline{y} .

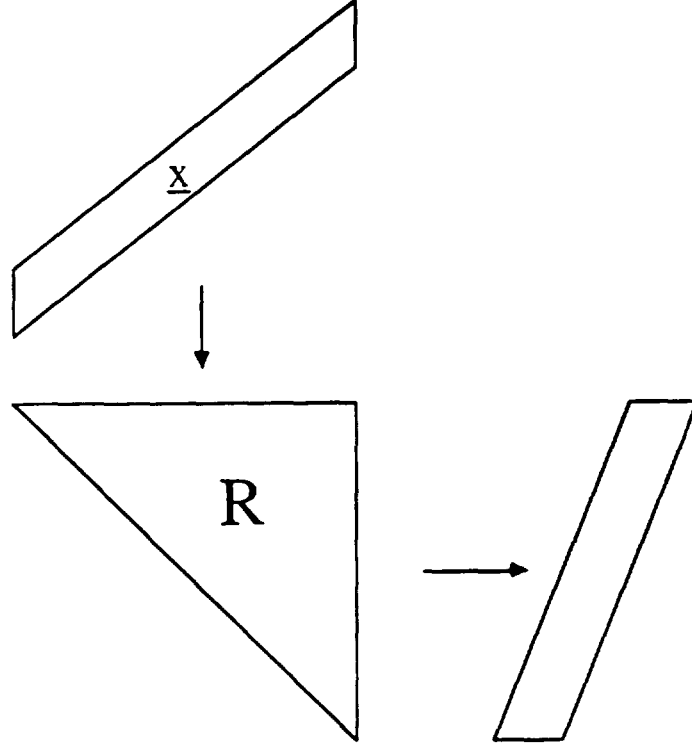


Figure 6.26: Computation of $R^{-T}\mathbf{x}$ using a triarray.

C. Triangular-Matrix Multiplication

The multiplication of a triangular matrix R and an rectangular full dense matrix B is given by

$$(C)_{ij} = (RB)_{ij} = \sum_{k=i}^n r_{ik} b_{kj}, \quad (6.205)$$

where r_{ik} and b_{kj} are elements of matrices R and B . Using the same array as in Fig. 6.25, with R prestored in the triarray and the operations shown in the third column of Table 6.5, this multiplication can be easily obtained if B is inputted row by row as in Fig. 6.27.

D. Matrix Multiplication

There are many ways to implement a full matrix-matrix multiplication in systolic array [52]. In Fig. 6.28, we show a typical architecture that can be incorporated with

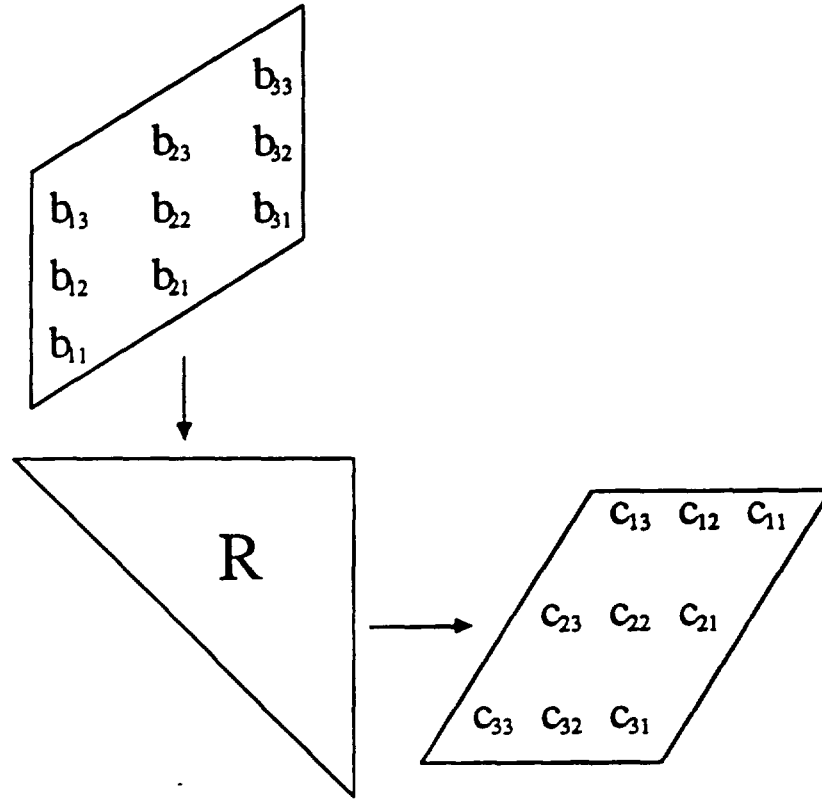


Figure 6.27: Multiplication of a triangular matrix and a full dense matrix.

the multi-phase operations to obtain eigenvectors. With input matrices Q and A arranged as in Fig. 6.28, the matrix B^T , where $B = AQ$, will sit in the rectangular array when the computation is completed. Details on this issue will be discussed in later sections.

6.3 QR Algorithm

In this section we review briefly the basic operation of the QR algorithm and show the evaluation of the eigenvectors from the cumulative multiplication of successive Q matrices. For a complex-valued $n \times n$ matrix A , it states that there is a unitary transform U such that $R = UAU^H$ is a upper triangular matrix with diagonal eigenvalues of descending order. This follows from the QR Algorithm [27, 92, 97] where

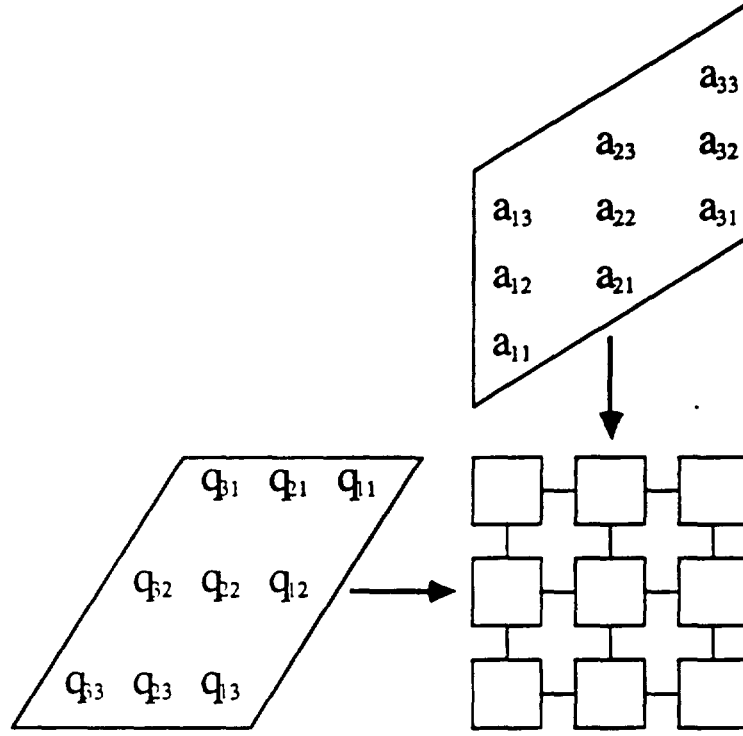


Figure 6.28: Matrix-matrix multiplication in a rectangular array.

by setting $A_1 = A$, we have $A_k = Q_k R_k$ and $A_{k+1} = R_k Q_k = Q_k^H A_k Q_k$, $k = 1, \dots$, with unitary Q_k and upper triangular R_k . Furthermore, A_k converges to the upper triangular matrix with diagonal eigenvalue elements. However, it is not obvious how to compute the eigenvectors from those Q_k and R_k we have calculated. With similar derivations as in [97], here we show how to obtain the eigenvector associated with the largest eigenvalue from cumulative multiplications of Q_k . From the above discussions, we have

$$A_{k+1} = Q_k^H Q_{k-1}^H \cdots Q_1^H A_1 Q_1 Q_2 \cdots Q_k. \quad (6.206)$$

Define

$$\tilde{Q}_k = \prod_{i=1}^k Q_i = Q_1 Q_2 \cdots Q_k,$$

$$\tilde{R}_k = \prod_{i=k}^1 R_i = R_k R_{k-1} \cdots R_1. \quad (6.207)$$

Then we have

$$\tilde{Q}_k A_{k+1} = A_1 \tilde{Q}_k. \quad (6.208)$$

Thus the multiplication of $\tilde{Q}_k \tilde{R}_k$ can be expressed as

$$\begin{aligned} \tilde{Q}_k \tilde{R}_k &= Q_1 Q_2 \cdots Q_k R_k R_{k-1} \cdots R_1 \\ &= \tilde{Q}_{k-1} A_k \tilde{R}_{k-1} = A_1 \tilde{Q}_{k-1} \tilde{R}_{k-1} = A_1^k = A^k. \end{aligned} \quad (6.209)$$

Let the eigenvalues of A satisfy, $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Denote the matrix eigenvectors and eigenvalues of A by X and Λ respectively. Then A^k is given by

$$A^k = X \Lambda^k X^{-1}. \quad (6.210)$$

Let the QR decomposition of X be $X = QR$ and the LU decomposition of X^{-1} be $X^{-1} = LU$, where L is an unit-lower triangular matrix. Then

$$A^k = QR \Lambda^k LU = QR (\Lambda^k L \Lambda^{-k}) \Lambda^k U, \quad (6.211)$$

where

$$\Lambda^k L \Lambda^{-k} = I + E_k, \quad (6.212)$$

and

$$(E_k)_{ij} = \begin{cases} l_{ij} (\lambda_i / \lambda_j)^k, & i > j, \\ 0, & \text{otherwise.} \end{cases} \quad (6.213)$$

Therefore, when k is large enough, we have $\lim_{k \rightarrow \infty} E_k = 0$ and thus $\Lambda^k L \Lambda^{-k}$ approaches the identity matrix. Then (6.211) can be rewritten as

$$A^k \rightarrow QR \Lambda^k U. \quad (6.214)$$

Since the term $R\Lambda^k U$ is an upper triangular matrix, comparing to (6.209) we can see that $\tilde{Q}_k \rightarrow Q$ when k is large. That is, the Q matrix of the QR decomposition of A^k approaches to that of the Q matrix of the QR decomposition of the matrix of eigenvector X . Define

$$\begin{aligned}\tilde{Q}_k &= [\tilde{q}_1, \tilde{q}_2, \dots, \tilde{q}_n], \\ X &= [\underline{x}_1, \underline{x}_2, \dots, \underline{x}_n],\end{aligned}\tag{6.215}$$

and r_{ij} as the (i, j) element of R . From $\tilde{Q}_k \rightarrow X$, we find $r_{11}\tilde{q}_1 \rightarrow \underline{x}_1$ when k is large. Since \underline{x}_1 is the eigenvector associated with the largest eigenvalue, we conclude that the first column of the matrix \tilde{Q}_k approach the eigenvector associated with the largest eigenvalue of matrix A when k is large. If the matrix A is symmetric, which is often the case for many signal processing applications, the similar transformation $A_{k+1} = \tilde{Q}_k^H A \tilde{Q}_k$ is also symmetric. Since A_{k+1} approaches the upper triangular matrix by the QR algorithm, A_{k+1} approaches a diagonal matrix. That is

$$A_k \rightarrow \Lambda,\tag{6.216}$$

and

$$\tilde{Q}_k \Lambda \rightarrow X.\tag{6.217}$$

In this case, for large k , the columns of \tilde{Q}_k become proportional to the columns of eigenvector in X .

If A is real, then A_k will converge to a real block upper triangular matrix with 1×1 and 2×2 main diagonal blocks. The complex conjugate pairs of eigenvectors of the 2×2 blocks can be solved easily using the quadratic formula. When A is not a square matrix, the singular values and vectors are of interest. For a $m \times n$ matrix B , where $m > n$, the SVD of B shows $B = U\Sigma V^T$, where U is a $m \times n$ matrix of

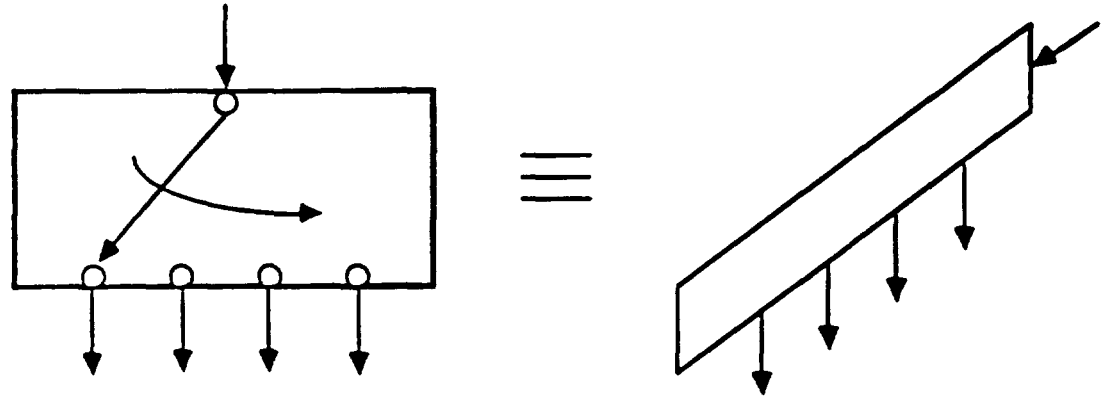


Figure 6.29: A circular multiplexer.

orthogonal columns, V is a $n \times n$ unitary matrix, and Σ is a $n \times n$ diagonal matrix with diagonal elements called singular values given in descending order. For most situations where high condition numbers are not encountered, a simple symmetric $n \times n$ matrix $C = B^T B$ can be formed and the matrix V can be found by direct use of the QR algorithm. Similarly, U can be found by using $D = B B^T$.

6.4 Multi-phase Systolic Algorithms

In this section, we introduced the multi-phase systolic algorithms to compute the QR algorithm. Two arrays, triangular and rectangular, can be used to compute the QR algorithm with some advantages and disadvantages for each. We shall show that our methods compute the Q matrix explicitly without requiring any global communication within the array. Before we consider the multi-phase algorithms, two communication switches are first discussed. A *circular multiplexer* is a device which takes its inputs and distributes them in different output positions as shown in Fig. 6.29. We use a skewed row to represent the circular multiplexer. A *first in/first out (FIFO) buffer*

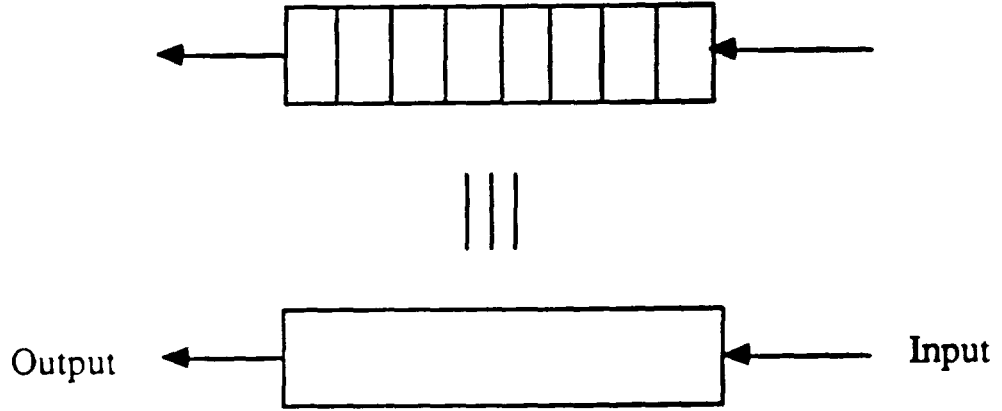


Figure 6.30: A first in/first out buffer.

is a buffer which takes its input to output in a first come first out manner as shown in Fig. 6.30. Both devices are commonly used in computer and microprocessor systems for data arrangement [36]. The computation of a QR algorithm consists of two basic steps. Initially, set $A_1 = A$.

- (1) for $k = 1, 2, \dots$, compute $A_k = Q_k R_k$;
- (2) compute $A_{k+1} = R_k Q_k$, stop if converge, otherwise go back to step (1).

6.4.1 Multi-phase Triangular Systolic Array

The QR Decomposition triarray proposed by Gentleman and Kung [26] is used in our approach. The R matrix is stored in the triarray after the computation. To compute the matrix A_{k+1} in step (2), the Q_k matrix has to be computed first. Let us call the computations in step (1) and step (2) an iteration. Several iterations are required for A_k to converge. For each iteration, we propose a three phase operation on a triarray as follows:

- **Phase 1:** QR decomposition for A_k

Compute the QR decomposition of the matrix $A_k = Q_k R_k$, with the upper triangular matrix R_k being stored in the triarray [29]. The data in A_k is inputted row by row skewed in time as shown in Fig. 6.31.

- **Phase 2:** Computing the Q_k matrix

From the QR Decomposition, we have $R_k^{-T} A_k^T = Q_k^T$. Let the i^{th} column of matrices A_k^T and Q_k^T be denoted by \underline{a}_i and \underline{q}_i respectively. Then

$$R_k^{-T} [\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n] = [\underline{q}_1, \underline{q}_2, \dots, \underline{q}_n]. \quad (6.218)$$

Section 6.2 showed that $R_k^{-T} \underline{x}$ can be computed in a triarray same as the one used in Phase 1. Since the i^{th} column of A_k^T is the i^{th} row of A_k , then with A_k inputted row by row skewed in time as shown in Fig. 6.32, the operations of the processing cells are given in the second column of Table 6.5. The triarray computes the Q_k matrix of A_k . The matrix Q_k is then outputted row by row as shown in Fig. 6.32. In order to start Phase 3, the matrix Q_k has to be in the form of Fig. 6.33. Observe that the output Q_k of phase 2 shares the same snap-shot order as the desired arrangement of Q_k in Phase 3 after a transpose operation. A circular multiplexer is used to distribute each column output of Q_k into row input as indicated in Fig. 6.32.

- **Phase 3:** Computing $R_k Q_k$

With the operations of the processing cell as shown in the third column of Table 6.5 and the Q_k obtained in Phase 2, Fig. 6.33 shows the computation of $A_{k+1} = R_k Q_k$ in the triarray. Then the matrix A_{k+1} comes out column by column from the right side of the triarray. Again, we observe that A_{k+1}

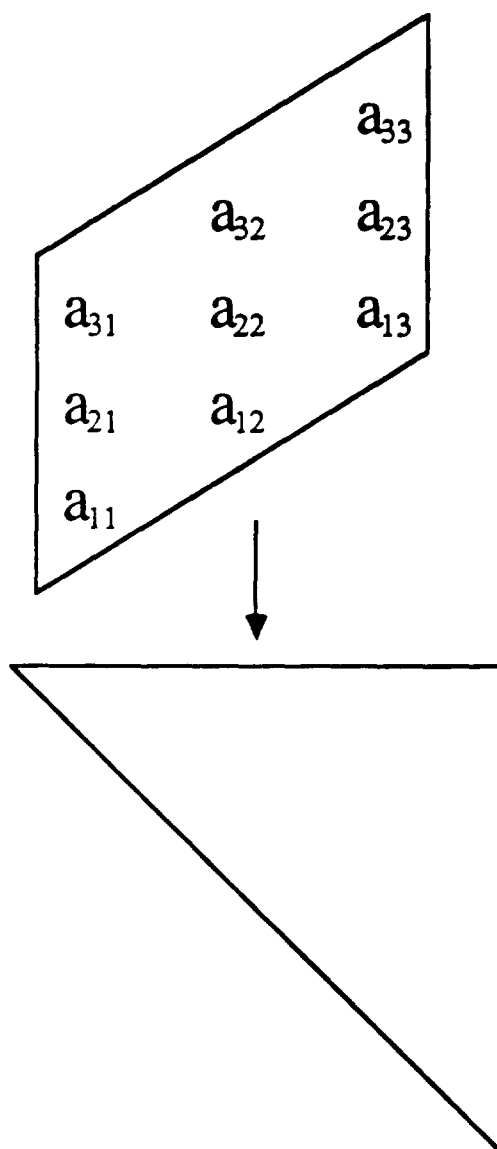


Figure 6.31: Phase 1: The QR decomposition.

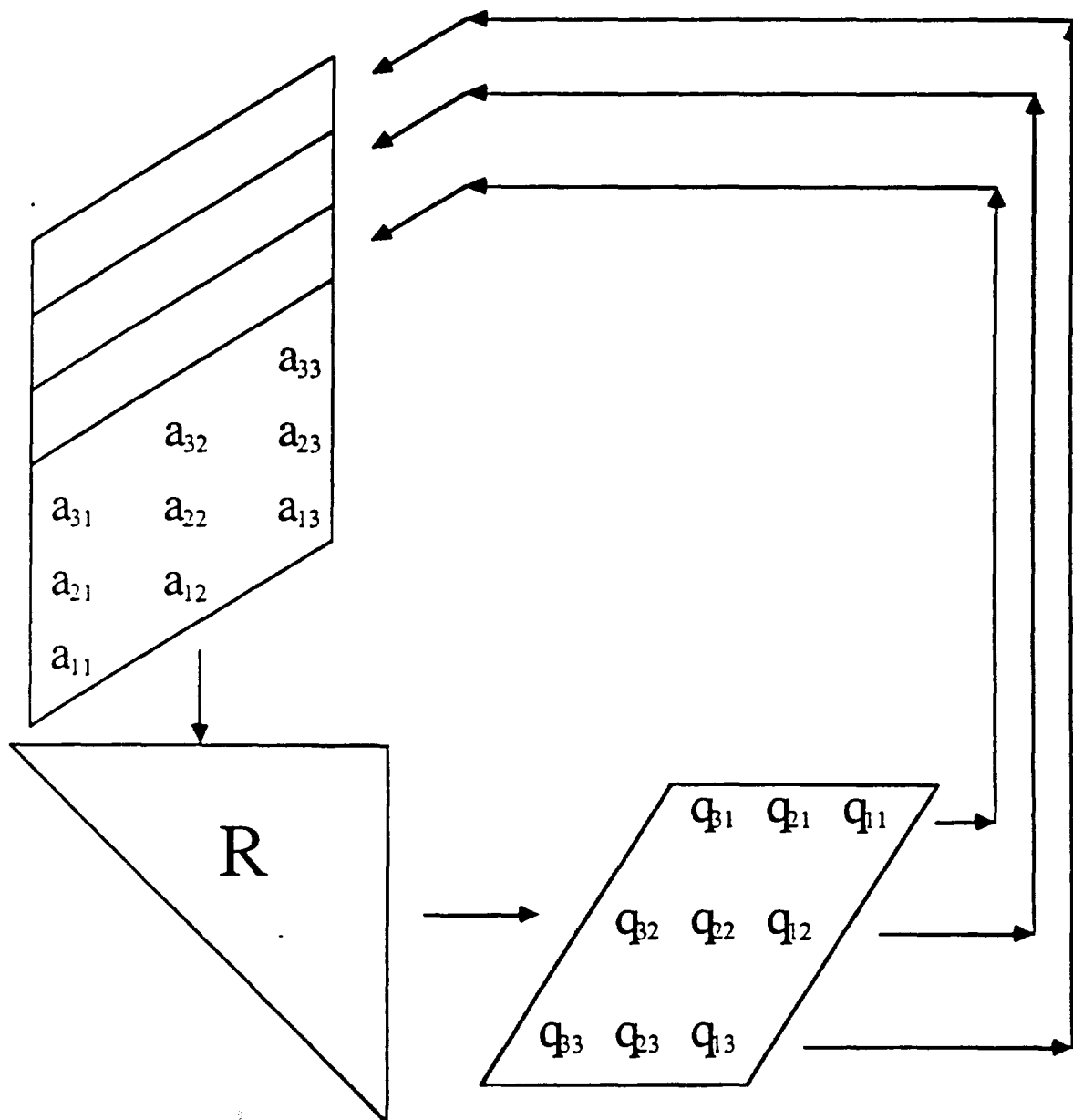


Figure 6.32: Phase 2: Computing the Q matrix.

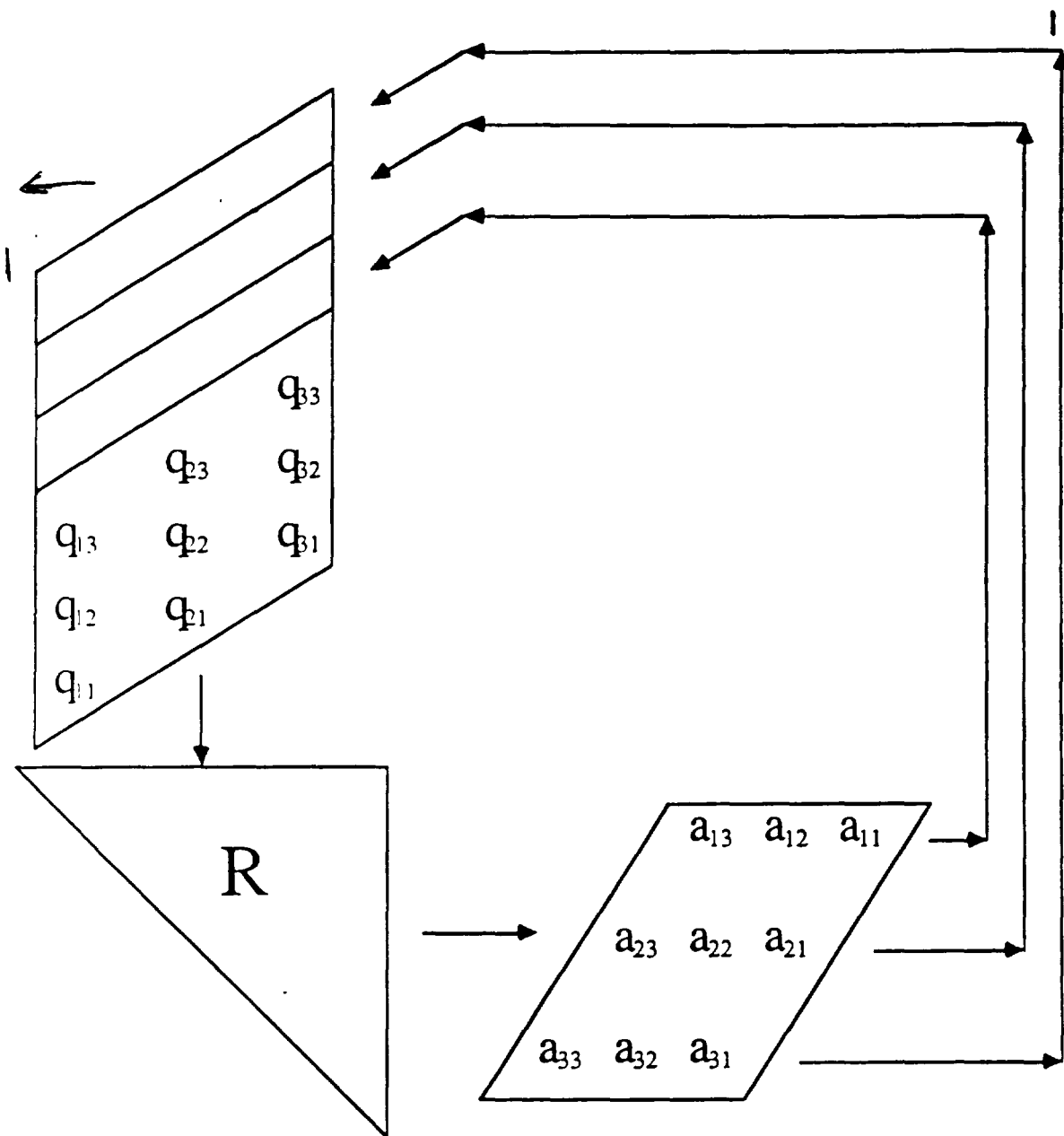


Figure 6.33: Phase 3: Computing the matrix product RQ .

shares the same snap-shot order as the desired arrangement of A_k in Phase 1 after a transpose operation. If not convergent, a new iteration is repeated by feeding back A_{k+1} into the triarray after using a circular multiplexer as shown in Fig. 6.33. Then Phase 1 operation begins as in Fig. 6.31.

An attractive property of this multi-phase operation is that the feedback requirement of the matrices in different phases are identical. Thus, only a circular multiplexer is needed for each row outside the array. Observe that each column of the matrices inputted in all of the phases need n time steps to process and the next phase can be started at time $n + 1$. We find once the data outputted at the right hand side of the triarray, after passing through the circular multiplexer, it can be piped into the array for the next phase computation without suffering any delay. If we assume the multiplexer is ideal such the delay in the multiplexer can be ignored, it takes $3n + (2n - 1) = 5n - 1$ system clocks for one iteration. The $(2n - 1)$ term represents the initial time to feed the data into the array. Suppose the number of iteration required for convergence is S , then the total number of system clocks needed is $3Sn + (2n - 1)$. Thus, the converge rate of this algorithm is of the order of $O((3S + 2)n)$. After the convergence of the A_k matrix, those values on the boundary cell are the eigenvalues of the A matrix.

6.4.2 Multi-phase Rectangular Systolic Array

The above method requires the use of the R^{-T} operation in the computations. From a numerical stability point of view, we may want to consider an alternative that uses a square matrix for cumulative multiplication of the rotation parameters. Fig. 6.34 shows a square matrix which is an extended version of the Gentleman-Kung's trian-

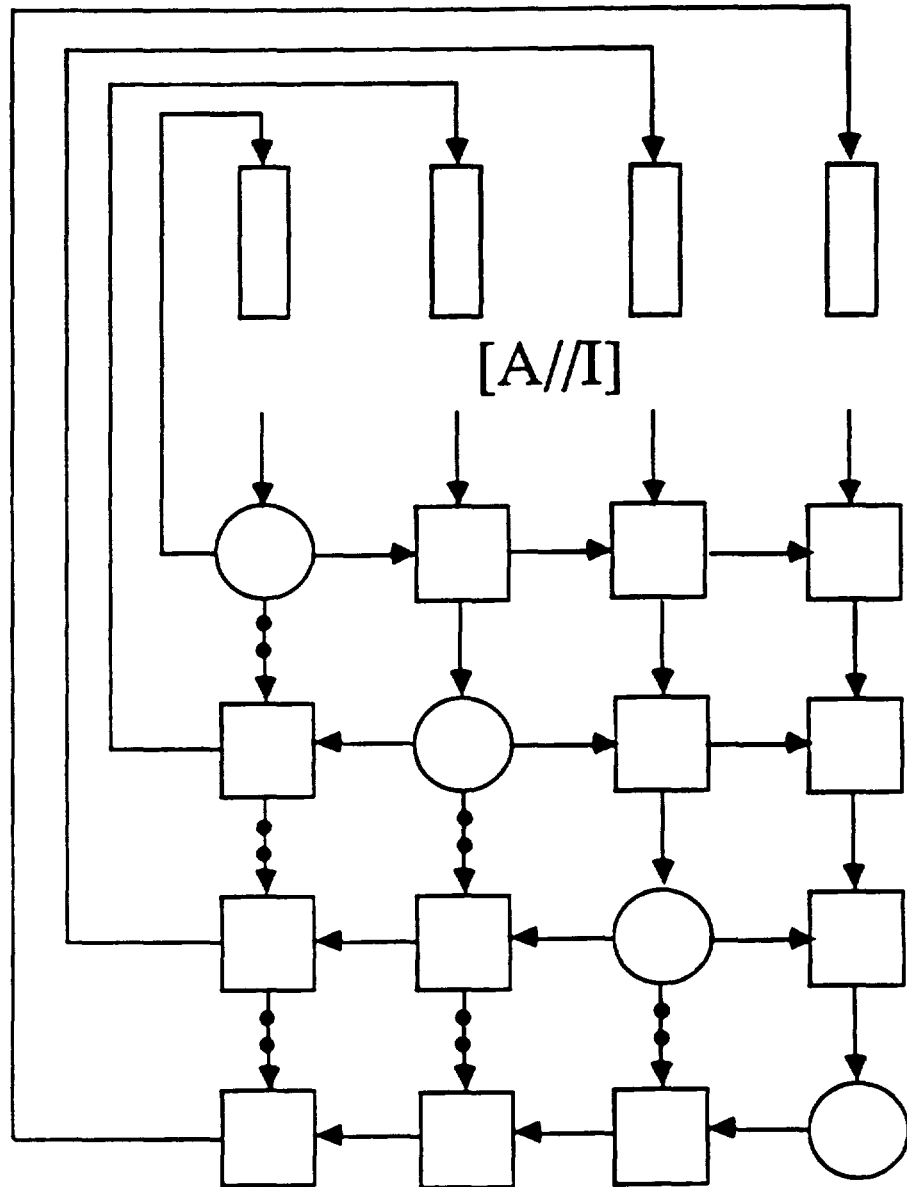


Figure 6.34: Multi-phase rectangular array for the QR iteration.

gular array with two delay elements (represented by black dots in Fig. 6.34) in the vertical communication links of the lower triangular part of the array. The processors in the lower triangular part are identical to the internal cells in the upper triangular part. Denote

$$[A//I] = [\underline{\alpha}_1; \underline{e}_1, \underline{\alpha}_2; \underline{e}_2, \dots, \underline{\alpha}_n; \underline{e}_n] \quad (6.219)$$

as the parallel combination of matrix A and I , where $A = [\underline{\alpha}_1, \underline{\alpha}_2, \dots, \underline{\alpha}_n]$ and I is a $n \times n$ identity matrix with \underline{e}_i as its i^{th} column. The square array takes the input $[A//I]$, while rotating A into an upper triangular matrix, I is used to cumulate the rotation parameters by

$$Q[A//I] = [R//Q]. \quad (6.220)$$

Noted that processors in the upper triangular part not only rotate the matrix A but also cumulatively multiply the rotation parameters with I . Thus, its work load is, in general, twice as that of Gentleman-Kung's internal cell. Processors in the lower part, on the other hand, only cumulate Q from the propagated rotation parameters. A two phases operation for QR iteration is proposed as follows:

- **Phase 1: QR decomposition**

Compute the QR decomposition of matrix $A_k = Q_k R_k$; both Q_k and R_k are obtained and stored. Then each row of Q_k is piped out and fed back to the array through a FIFO buffer as shown in Fig. 6.34.

- **Phase 2: Computing $R_k Q_k$**

In this phase, the operation is identical to that of the Phase 3 in the triangular array. A circular multiplexer is used to transform A_{k+1} from row output into column input. Continue this iteration until converged.

Due to the delay elements at the lower triangular part of the array and the work load of each processor (except those in lower triangular part) is twice as that of triangular array, the time to obtain the i^{th} row of the Q_k matrix, t_i , is

$$t_i = \max(2(n + 3i - 3), 2(2n + i - 2)),$$

where $2(n + 3i - 3)$ is the time for the left-most cell of the i^{th} row to obtain its Q element and $2(2n + i - 2)$ is the time for the right-most cell to finish. Obviously, when $i \geq \lceil \frac{n+1}{2} \rceil$, $t_i = 2(n + 3i - 3)$. Thus, the time required to obtain the whole Q matrix is $t_n = 8n - 6$. By assuming that it takes time n to sequentially pipe out the Q_k matrix, this algorithm takes $(9n - 6) + n + (2n - 1)$ to complete an iteration in the worst cast. Again, denote the number of iteration as S , this algorithm converge in the order of $O(S(10n - 6) + 2n - 1) = O((10S + 2)n)$. Of course, the performance can be improved by piping out each row of Q matrix when it is available instead of waiting for the whole Q matrix is available. With this, the performance can reach to the order of $O((9S + 2)n)$.

6.4.3 The Hessenberg Reduction

In order to perform the QR algorithm efficiently in conventional Von Neumann type series computers, we usually transform the data matrix A into an upper Hessenberg matrix before the QR iteration is started. With this transformation, the amount of work per iteration is reduced from $O(n^3)$ to $O(n^2)$ [30]. However, this argument may not be true for parallel processing architectures. The reasons are two folds:

1. Due to the hardware resource in a parallel processing architecture, the computations can be performed concurrently without hindering the processing time.

For example, the computation times of the two multi-phase arrays discussed above are of the order $O(n)$.

2. The data matrix is usually not of the Hessenberg form. The pre-processing of the data matrix to Hessenberg form may not be able to be incorporated with the following computations. That is, the pre-processing must be done separately.

Both reasons may lead to the conclusion that the Hessenberg form is not of interest and practical for parallel processing of the QR algorithm unless the Hessenberg form can be obtained easily by using the same parallel processing architecture. Many papers prevented to answer this question by assuming the Hessenberg form (or the tridiagonal form) is already available. Fortunately, the Hessenberg form can be obtained easily as an additional part of the multi-phase operations.

To obtain the Hessenberg form, we can choose an unitary similarity transformation U such that $A_1 = U^H A U$ is a upper Hessenberg matrix [30]. The transformation U can be obtained from sequences of Givens rotations. Denote G_i as the product of the Givens rotation matrices which zero out the proper positions of the i^{th} column. Since the first i^{th} rows will not be affected by G_i , the matrix G_i is of the form $G_i = \text{diag}(I_i, \bar{G}_i)$, where I_i is an identity matrix of dimension i . Suppose the Hessenberg form through its first $k - 1$ columns have been obtained

$$(G_1 \cdots G_{k-1})^H A (G_1 \cdots G_{k-1}) = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \end{bmatrix}, \quad (6.221)$$

where B_{11} and B_{33} are $(k-1) \times (k-1)$ and $(n-k) \times (n-k)$ matrices respectively.

Then

$$(G_1 \cdots G_k)^H A (G_1 \cdots G_k) = \begin{bmatrix} B_{11} & B_{12} & B_{13}\bar{G}_k \\ B_{21} & B_{22} & B_{23}\bar{G}_k \\ 0 & \bar{G}_k^H B_{32} & \bar{G}_k^H B_{33}\bar{G}_k \end{bmatrix} \quad (6.222)$$

is a Hessenberg form through its first k columns. Thus

$$A_1 = (G_1 \cdots G_{n-1})^H A (G_1 \cdots G_{n-1}) \quad (6.223)$$

is an upper Hessenberg matrix and

$$U = G_1 \cdots G_{n-1} = \begin{bmatrix} 1 & \underline{0}^T \\ \underline{0} & G \end{bmatrix}. \quad (6.224)$$

Denote

$$A = \begin{bmatrix} \underline{a}^T \\ \bar{A} \end{bmatrix} = U \tilde{A} = \begin{bmatrix} 1 & \underline{0}^T \\ \underline{0} & G \end{bmatrix} \cdot \begin{bmatrix} \underline{a}^T \\ \tilde{R} \end{bmatrix}, \quad (6.225)$$

where $\bar{A} = G\tilde{R}$ and \tilde{R} is of the form of an upper triangular matrix without the lowest vertex element. Then $A_1 = \tilde{A}U = U^H AU$. Obviously, this is similar to the computations in the QR iteration. To obtain G and \tilde{R} , let

$$\hat{A} = \begin{bmatrix} \bar{A} \\ - - - - - \\ 0, 0, \dots, 1 \end{bmatrix}. \quad (6.226)$$

The QRD of \hat{A} is

$$\hat{A} = \hat{Q}\hat{R} = \begin{bmatrix} G & \underline{0}^T \\ \underline{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{R} \\ 0, 0, \dots, 1 \end{bmatrix}. \quad (6.227)$$

Now, we can use the multi-phase operations to obtain the upper Hessenberg form.

We call this the Phase 0 operation.

Phase 0: The Hessenberg Reduction.

(1) Phase 1 Operation: QRD of \hat{A} .

from $\hat{A} = \hat{Q}\hat{R}$, we obtain \tilde{R} in the triarray.

(2) Phase 2 Operation: Computing G matrix.

From $\hat{Q} = \hat{R}^{-T}\hat{A}^T$, we obtain matrix G .

(3) Phase 3 Operation: Computing the Hessenberg matrix A_1 .

By forming

$$\tilde{A} = \begin{bmatrix} \underline{a}^T \\ \tilde{R} \end{bmatrix}$$

and

$$U = \begin{bmatrix} 1 & \underline{0}^T \\ \underline{0} & G \end{bmatrix},$$

we obtain the Hessenberg matrix $A_1 = \tilde{A}U$.

6.4.4 Computing the Eigenvectors

To compute an eigenvector, a matrix multiplication systolic array can be incorporated with the multi-phase array such that those matrices Q_1, \dots, Q_k are cumulated to form the \tilde{Q}_k matrix. Noted that $\tilde{Q}_k = \tilde{Q}_{k-1}Q_k$ and the matrix \tilde{Q}_{k-1} is available at the start of the k^{th} iteration, while the matrix Q_k coming out at Phase 2 operation of the k^{th} iteration. Then \tilde{Q}_k is obtained by multiplying \tilde{Q}_{k-1} and Q_k as shown in Fig. 6.28. A system configuration for triangular array is shown in Fig. 6.35 and that for rectangular is shown in Fig. 6.36. As discussed in section 6.3, for a symmetric A matrix, when A_k converged, \tilde{Q}_k yields the matrix of eigenvectors. For a non-symmetric A matrix, the first column of \tilde{Q}_k yields the eigenvector associated with the largest eigenvalue.

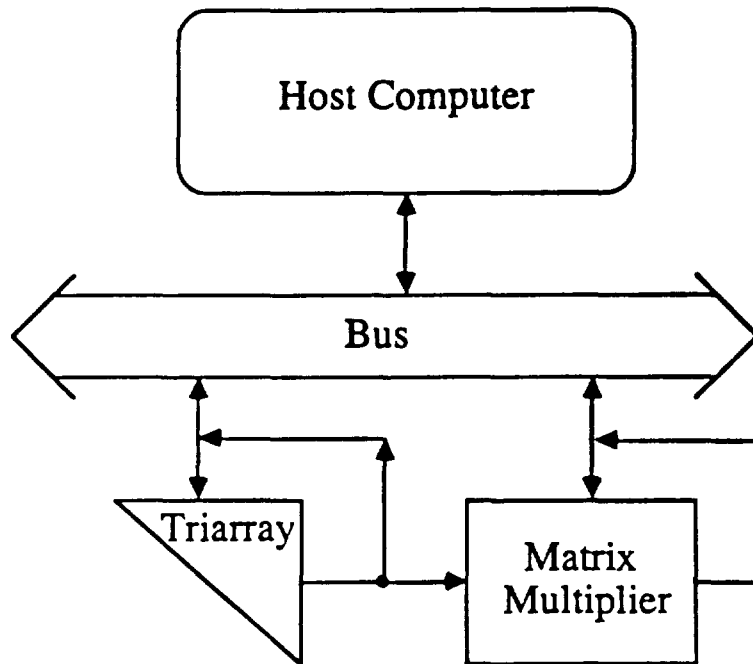


Figure 6.35: System configuration of the multi-phase triarray.

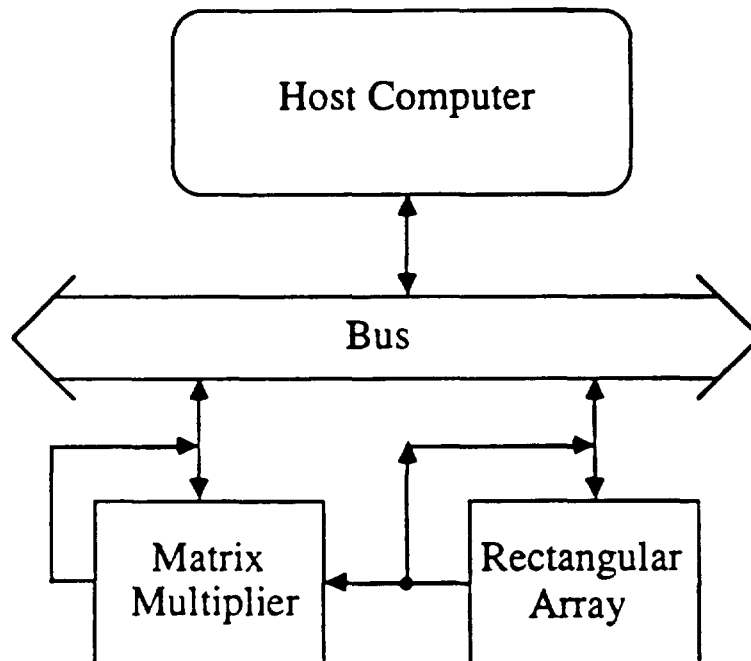


Figure 6.36: System configuration of the multi-phase rectangular array.

	Triangular array	Rectangular array
Computation time	$O((3S+2)n)$	$O((10S+2)n)$ worst case
Numerical stability	fair	stable
Number of cells	$\frac{n(n+1)}{2}$	n^2 plus ($n^2 - n$) d-elements
I/O ports	$2n$	$3n$
Utilization	1	<1
Communication devices	1	2

Table 6.7: Comparisons of the multi-phase triarray and rectangular array.

6.5 Performance Efficiency

6.5.1 Comparisons of the arrays

Although there are three phases of operations, the arithemtical operations in Phase 2 and Phase 3 form a subset of the operations executed in Phase 1. Therefore we do not increase the cell complexity in the multi-phase arrays. The performance and characteristics of both triangular and rectangular arrays considered above are summarized in Table 6.7. The advantages of the triangular array are: it has less computational time as well as less number of cells, I/O ports, and communication devices. Furthermore, all the the processing cells are fully utilized. However, due to the computation of R^{-T} in Phase 2 of the operations, it may be numerical unstable for certain highly

ill-conditioning data. For example, consider the matrix given by

$$\begin{bmatrix} 0.7601 & -0.3967 & 0.6060 \\ -0.3967 & 1.7475 & -0.1962 \\ 0.6060 & -0.1962 & 0.4924 \end{bmatrix}$$

with eigenvalues $\{2.0, 1.0, 10^{-12}\}$. If the triarray algorithm, which uses R^{-T} to obtain Q^T , is used, the eigenvalues are obtained as $\{2.0, 1.0, 3.6818 \cdot 10^{-6}\}$. On the other hand, eigenvalues are obtained as $\{2.0, 1.0, 9.9999 \cdot 10^{-13}\}$ if the R^{-T} is not explicitly computed. As a result, we have a complexity versus numerical stability tradeoff for the two multi-phase arrays.

6.5.2 Rate of Convergence

Similar to Luk in [72], by convergence we mean that the parameter $off(A_k)$ defined as

$$off(A_k) = \frac{\sum_{i < j} a_{ij}^2(k)}{N}, \quad (6.228)$$

where N is the number of off-diagonal elements, has fallen below some prechosen tolerance value. As indicated in [72], it is difficult to monitor $off(A_k)$ in the parallel computation. Luk then proposed that the iteration be stopped after a sufficiently large number S of iterations. In the studies of Brent and Luk [12, 72], they found that $S \leq 9$ for random symmetric matrices of order $n \leq 230$ and $S \leq 6$ for $n \leq 24$. Therefore, they chosen $S = 10$ for $n \leq 100$ for Jacobi-like method. Similar to their approach, we apply the QR algorithm to random $n \times n$ symmetric matrices (a_{ij}) , where the elements a_{ij} for $1 \leq i \leq j \leq n$ were uniformly and independently distributed in $[-1, 1]$. The tolerance to meet the stopping condition is $off(A_k) \leq 10^{-10}$. We can see from Fig. 6.37 that the number of iterations for a QR algorithm to converge is in

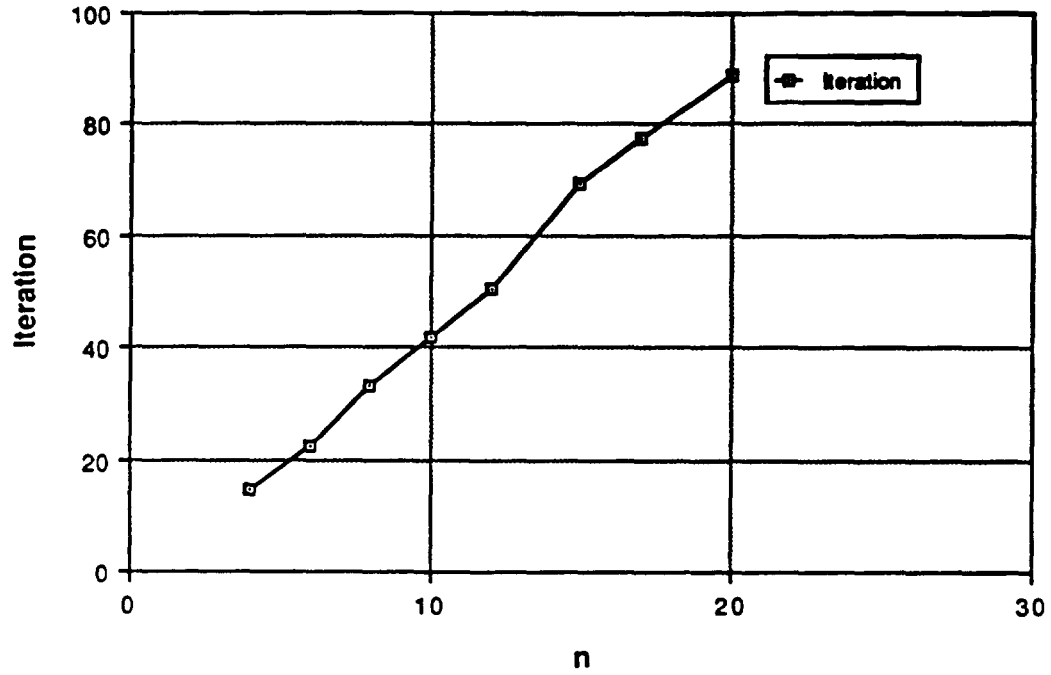


Figure 6.37: The number of iterations for a QR algorithm to converge versus the matrix size.

the order of 10 for matrix size smaller than 20×20 . Even though we can reduce the matrix to Hessenberg form for full dense matrix or tridiagonal form for symmetric matrix, and the QR iteration with origin shift can accelerated the convergent rate [32, 91, 98], the number of iteration is still in the order of 10. As an example, the 4×4 tridiagonal matrix

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 3 & 4 & 0 \\ 0 & 4 & 5 & 6 \\ 0 & 0 & 6 & 7 \end{bmatrix}$$

still requires eight iterations to converge when the most elegant symmetric QR algorithm, the Symmetrix QR Algorithm, was used [27, pp.424] This kind of property is not desirable for parallel processing implementation. It is known that Jacobi-like

method may require more flops as compared to the symmetric QR algorithm. However, due to parallel implementation, many rotations may take place at the same time. The computations involved in QR algorithm and Jacobi-like method are generally of the same complexity. From these discussions, the one which requires less number of iterations is more attractive from the parallel implementational point of view. Furthermore, the convergence rate of an QR iteration depends on the ratio of the eigenvalues. In our simulations, in more than 10% of the cases, the randomly generated symmetric matrices required significantly larger number of iterations to converge. As pointed out before, it is difficult to monitor the quantity $off(A_k)$ to decide when the algorithm converges in the parallel computation. Since the convergence rate is highly dependent on the ratio of eigenvalues, there is no general rule to choose a sufficient number of iteration S like Jacobi-like method to insure convergence. This is also an undesirable intrinsic property of the QR algorithm for parallel implementation as compared to that of the Jacobi-like method.

6.6 Efficient Fault-tolerance Schemes

Reliable implementation is quite essential in parallel processing architectures. For a complex parallel processing system, a single fault from any part of the system can make the whole system useless. For various critical applications using spectral decomposition, highly-reliable computations are demanded. Fault-tolerance is therefore needed in many of these problems. A simple and cost effective fault-tolerant scheme is the checksum and weighted checksum proposed by Abraham et al [2]. This scheme is one of the typical examples of the algorithm-based fault-tolerance which has been applied to various signal processing and linear algebra operations [65]. Define

the checksum vector $\underline{e}^T = [1, 1, \dots, 1]$, the column, row and full checksum matrices A_c , A_r and A_f of a square n -by- n matrix A are defined as

$$A_c = \begin{bmatrix} A \\ \underline{e}^T A \end{bmatrix},$$

$$A_r = \begin{bmatrix} A & A\underline{e} \end{bmatrix},$$

$$A_f = \begin{bmatrix} A & A\underline{e} \\ \underline{e}^T A & \underline{e}^T A\underline{e} \end{bmatrix}.$$

If there is any fault occurring during the computation, the checksum criterion is not met and thus the fault is detected. The weighted checksum scheme can be further used to correct errors [47]. It has been suggested in [17] that checksum/weighted checksum scheme can be incorporated into the QR iteration for error detection. These properties as well as others are considered here for the multi-phase arrays.

Since there are different operations in different phases, the inherent natures of the operations of each phase is thus different and should be examined for possible fault-tolerant implementation individually. The fault-tolerant schemes for each phase of the multi-phase triangular array are given as follows:

- **Phase 1:** As pointed out in [17, 65], row checksum is invariant for the QR decomposition. It can be seen

$$A_r = \begin{bmatrix} A & A\underline{e} \end{bmatrix} = Q \begin{bmatrix} R & R\underline{e} \end{bmatrix} = QR_r. \quad (6.229)$$

This means that the QR decomposition of a row checksum matrix results in a row checksum upper triangular matrix. Fig. 6.38 shows the implementation of this scheme.

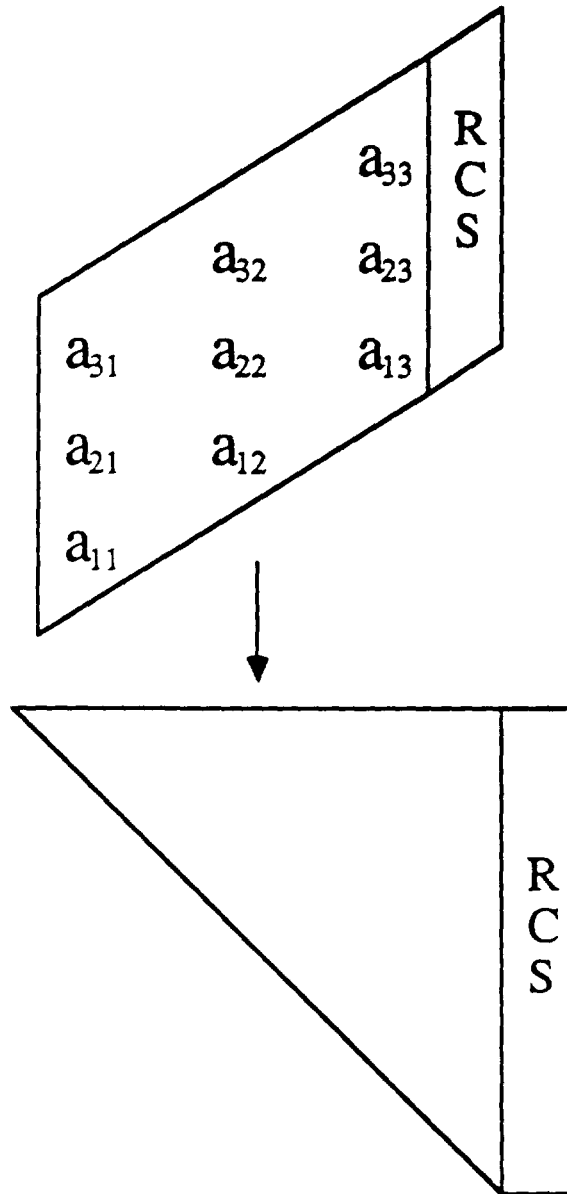


Figure 6.38: Row checksum for $A_r = QR_r$.

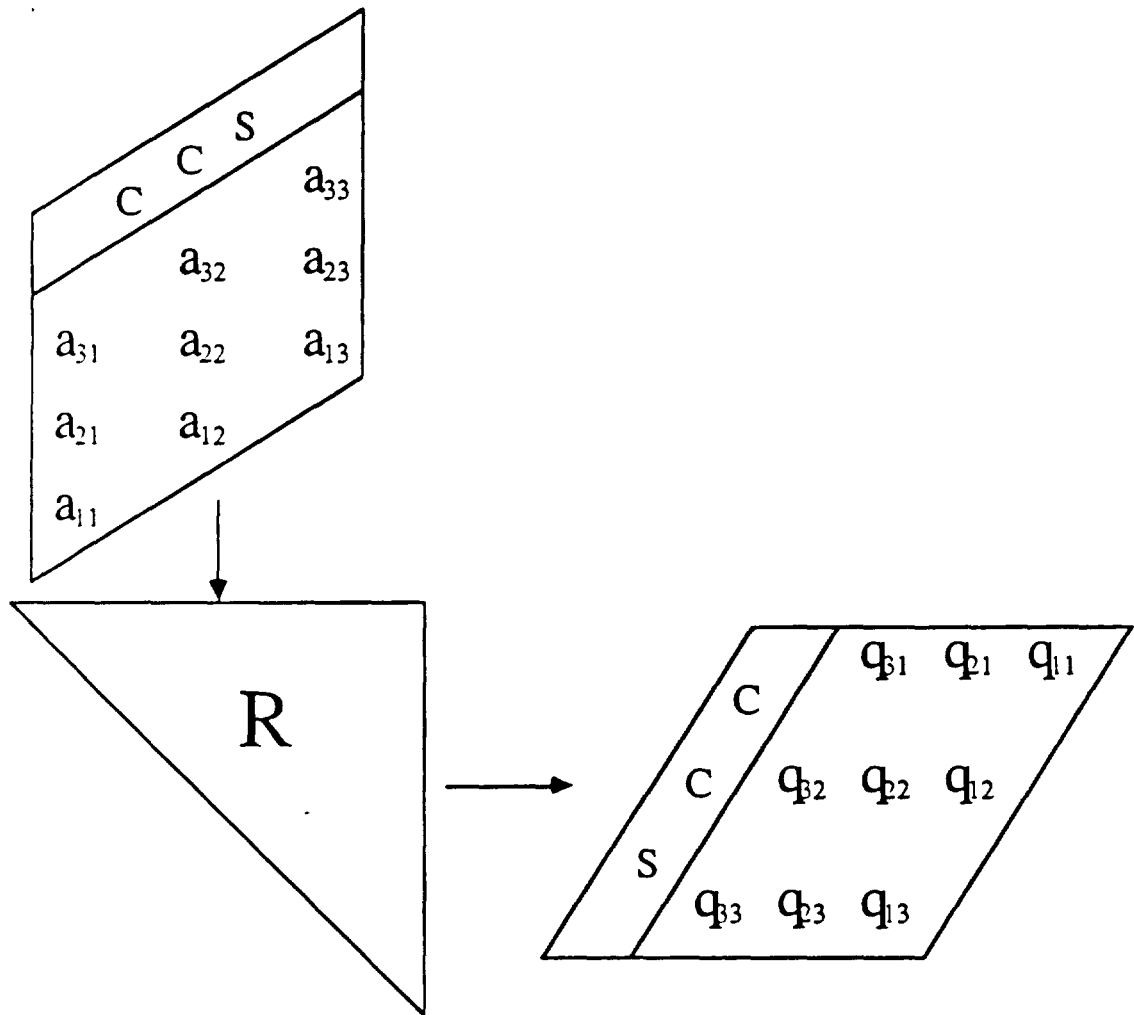


Figure 6.39: Column checksum for $R^{-T} A_c^T = Q_c^T$.

- **Phase 2:** Due to the nature of computations in this phase, row checksum is no longer valid. Fortunately, column checksum is possible as given by

$$R^{-T}A_c^T = R^{-T} \begin{bmatrix} A^T & A^T \underline{e} \end{bmatrix} = \begin{bmatrix} Q^T & Q^T \underline{e} \end{bmatrix} = Q_c^T. \quad (6.230)$$

An implementation of this scheme is shown in Fig. 6.39.

- **Phase 3:** Although a row checksum upper triangular matrix R_r and a column checksum unitary matrix Q_c are obtained in the above phases, unfortunately, $R_r Q_c$ does not yield any relevant use. By defining the *trace* operation as the sum of the diagonal elements in a square matrix, we obtain $Tr[AB] = Tr[BA]$, where A and B are square matrices. Therefore,

$$Tr[A_{k+1}] = Tr[R_k Q_k] = Tr[Q_k R_k] = Tr[A_k] = \sum_{i=1}^n \lambda_i, \quad (6.231)$$

where λ_i is the eigenvalue of matrix A_k . This invariant property can be used to check the result of the Phase 3 operation. Once the trace of A_k is different from the trace obtained before, a fault is then detected during the Phase 3 computation.

For the rectangular array, the Phase 2 operation is the same as the Phase 3 operation of the triangular array. For its Phase 1 operation, an interesting feature of this computation is given by

$$Q[A_r // I_r] = [R_r // Q_r]. \quad (6.232)$$

That is, a row checksum of the parallel combination of matrices A and I gives a row checksum of the upper triangular matrix R_r and a row checksum of the unitary matrix Q_r .

Chapter 7

Conclusions and Future Research

This dissertation has addressed various aspects of the RLS problems and spectral decomposition algorithms based on QRD approach.

In Chapter 2, we have shown that the Householder transformation can be implemented on a systolic array. By using a two-level pipelined implementation, the throughput of the SBHT RLS systolic array can be made as fast as that of the original Givens array in [78]. While the system latency is longer for the SBHT, it provides a better numerical stability than the Givens method. Clearly, the Givens array is a special case of the SBHT array with a block size of one. In general, the block size is an important variable. A larger block size results in a better numerical stability, while the system latency is increased. Many known properties of the Givens array are also applicable to the SBHT array. For example, the real-time algorithm-based fault-tolerant scheme proposed in [65] can also be easily incorporated into the SBHT RLS array. From the results described in Chapter 2, it appears that the Householder transformation method is useful in real-time high throughput applications of modern signal processing as well as in VLSI implementation.

In Chapter 3, special inherent natures of a QRD recursive LS systolic array are used to design a real-time, low-cost algorithm-based fault-tolerant system. By proper design of the artificial desired response, the residual output serves as a concurrent error detector. The disadvantages of the CSE scheme to detect fault are thus prevented. The proposed method requires the same complexity as that of the CSE scheme in [17] without hindering the throughput rate of the system. Two methods, FFL and CSE, are then introduced to locate the faulty row. For both methods, the recovery latency is achieved in $O(p)$ time. However, the recovery latency of the CSE method is generally less than that of the FFL method in that parallel execution is possible and multiple ports can be accessed. Once the faulty row is determined, an order-degraded reconfiguration is performed so that the system is operating in an gracefully degraded manner. Any single fault occurring in the system will not cause a catastrophic degradation of the system and thus one critical requirement for many high performance and demanding VLSI signal processing tasks is met.

This fault-tolerance system is general in the sense that it is independent of the implementation algorithms. It is applicable to Given rotation method as well as those methods such as Modified Gram-Schmidt, square-root free Given methods [19], and Householder transformations [66] etc. The LS problems with constraints such as MVDR beamforming is also applicable [80]. The residual method for concurrent error detection is robust in the sense that the probability of error detection given by a fault occurred equals one (for infinite-precision implementation). This scheme could have great impact on next generation of radar systems which may use VLSI array processors as their central signal processing units.

In Chapter 4, We present an important observation that rotation parameters of the

QRD LS algorithm will eventually reach the *quasi steady-state*. With this observation, the dynamic range of each processing cell is then derived to insure correct operation of the algorithm. Also, we can prove the stability of the QRD LS algorithm under finite-precision implementation with this observation. The missing error detection and false alarm problems are considered. We present a design of the memory size which is overflow free without missing error detection and false alarm problems.

In Chapter 5, the order degraded performance is derived and a geometric interpretation is given. A scheme to estimate the optimal residual under faulty situation is also presented.

The multi-phase systolic algorithms proposed in Chapter 6 can be used efficiently to solve the eigenvalue and SVD problems based on the QR algorithm. In particular, the eigenvectors can be obtained without global communication within the array using the multi-phase operations. We showed that the QR algorithm can achieve a parallel implementation on a single architecture. Two systolic arrays, a triangular and a rectangular, are proposed for multi-phase implementation. Efficient algorithm-based fault-tolerance schemes can be incorporated with both arrays easily. Since the operations in each phase belong to the same types of computations, the cell complexity is thus not increased by multi-phase operations. There is a tradeoff between numerical stability and complexity for both arrays. Each iteration takes $O(n)$ time unit while the time required for convergence is $O(Sn)$, where S is the number of iterations. Unlike the Jacobi-like method, the convergence rate of the QR algorithm depends on the ratio of the eigenvalues. As a consequence, S may vary differently for the matrix of the same size, with or without origin shift to accelerate the convergence. Generally, S is in the order of 10 for the QR algorithm. While we have demonstrated

the advantage of the QR algorithm that can yield two multi-phase systolic algorithms implementable on single architectures without requiring global connections, the intrinsic convergence rate for the QR algorithm makes it less attractive as compared to the Jacobi-like method in parallel implementation. Depending on specific system and hardware requirements, one approach may be more desirable than the other. Of course, it is most meaningful to have two basic approaches to choose from for real-time VLSI signal processing based on spectral decomposition.

There are various continuing problems extended from this dissertation that can be done in the future. For the SBHT RLS systolic algorithm, to pursue a possible CORDIC implementation would be an interesting problem. An algorithm-based fault-tolerance has been proposed for the QRD RLS systolic algorithm. It would be a valid question to ask if there is any efficient fault-tolerant scheme for the transversal or lattice algorithms. Different from the residual estimation under faulty situation, it would be possible for us to ask if we can estimate the full-order optimal residual under the order degraded operation. Also, there are still some questions on re-convergence of a fault-tolerant QRD RLS systolic array. When there is a fault occurred in the system, if it is a transient fault, how long will this fault affect the system? In another words, how soon will the adaptive system recover itself from the faulty situation and converge. There are two questions. The first one is how long will it recover from the short time transient fault without any cure? Does the position of the faulty processor play an important role? The next question is when we switch to order degraded operation, do we want to reinitialize the system or to restart from the contaminated contents?

Future research in the VLSI signal processing can be divided into the following

categories:

1. Algorithms and architectures aspects

To reconsider and to develop new parallel and pipelined algorithms, to design dedicated application-specific architectures for parallel algorithms (either in VLSI or DSP chip sets), and to find efficient fault-tolerant schemes for special-purposed applications are the major issues in this category.

2. Software Supported CAD Aspects

To provide simulation tools for algorithms and architectures development and to support automatic design of complicated systems are the major concerns of this category.

3. Application and Practical Implementation Aspects

There is a large application spectrum. To name a few, the areas such as sonar/radar, communication, image processing, HDTV, coding, video, speech processing, and music are of potential interest to researcher in the VLSI signal processing.

Bibliography

- [1] A.H. Abdallah and Y.H. Hu, "Parallel VLSI computing array implementation for signal subspace updating algorithm", IEEE Trans. ASSP, Vol 37, pp.742-748, 1989.
- [2] J.A. Abraham et al., "Fault tolerance techniques for systolic array", IEEE Computer, Vol 20, pp.65-76, July 1987.
- [3] C.J. Anfinson and F.T. Luk, "A linear algebraic model of algorithm-based fault tolerance", Proc. IEEE Int'l Conf. Systolic Array, pp.483-493, May 1988.
- [4] C.J. Anfinson et al., "Algorithm-based fault-tolerant techniques for MVDR beam-forming", Proc. IEEE ICASSP, pp.2417-2420, May, 1989.
- [5] M. Annaratone *et al.*, "The Warp computer: Architecture, implementation, and performance", IEEE Trans. Computer, C-36, pp.1523-1538, Dec. 1987.
- [6] A. Avizienis, "Fault tolerant computing - An overview", IEEE Computer, Vol 4, pp.5, Jan. 1971.
- [7] P. Banerjee and J.A. Abraham, "Bounds on algorithm-based fault-tolerance in multiple processor systems", IEEE Trans. Computer, C-35, pp.296-306, 1986.
- [8] M.G. Bellanger, "Computational complexity and accuracy issues in fast least squares algorithms for adaptive filtering", Proc. IEEE ISCAS, pp.2635-2639, Finland, 1988.
- [9] G. Bienvenue and H.F. Mermoz, "New principle of array processing in underwater passive listening", in **VLSI and Modern Signal Processing**, S.Y. Kung et al., Eds., Prentice-Hall, 1985.
- [10] A.W. Bojanczyk, R.P. Brent, and F.R. de Hoog, "Parallel QR decomposition of Toeplitz matrices", Proc. SPIE Advanced Algorithms and Architectures for Signal Processing, pp.39-44, Aug. 1986.
- [11] A.W. Bojanczyk and A.O. Steinhardt, "Stabilized hyperbolic Householder transformations", IEEE Trans. ASSP, Vol 37, pp.1286-1288, Aug. 1989.

- [12] R.P. Brent and F.T. Luk, "The solution of singular-value singular-value and symmetric eigenvalue problems problems problems on multiprocessor array," SIAM J. Sci. Stat. Comput., Vol 6, pp. 69-84, Jan. 1985.
- [13] R.P. Brent and F.T. Luk, "A systolic array for the linear-time solution of Toeplitz systems of equations", Journal of VLSI and Computer Systems, 1, pp.1-22, 1985.
- [14] K. Bromley and J.M. Speiser, "Signal processing algorithm, architectures, and applications", Proc. SPIE, Vol. 431, pp. 2-6, 1983.
- [15] S.-W. Chan and C.-L. Wey, "The design of concurrent error diagnosable systolic arrays for band matrix multiplications", IEEE Trans. CAD, Vol 7, pp.21-37, Jan. 1988.
- [16] M. Chean and J.A.B. Fortes, "A taxonomy of reconfiguration techniques for fault-tolerant processor arrays", IEEE Computer, Vol. 23, pp.55-69, Jan. 1990.
- [17] C.-Y. Chen and J.A. Abraham, "Fault-tolerant systems for the computation of eigenvalues and singular values", Proc. SPIE, Vol 696, Advanced Algorithms and Architectures for Signal Processing, 1986.
- [18] C.-Y. Chen and J.A. Abraham, "Current error detection in VLSI processor arrays", Proc. SPIE, Vol 826, Advanced Algorithms and Architectures for Signal Processing II, pp. 205-214, 1987.
- [19] M.J. Chen, "On realizations and performances of least-squares estimation and Kalman filtering by systolic array", Ph.D. dissertation, Electrical Engineering Dept., UCLA, 1987.
- [20] S.I. Chou and C.M. Rader, "Algorithm-based error detection of a Cholesky factor updating systolic array using CORDIC processors", Proc. SPIE, Real-time Signal Processing XI, pp.104-111, Aug. 1988.
- [21] J.M. Cioffi, "Limited-precision effects in adaptive filtering", IEEE Trans. CAS, VOL CAS-34, pp.821-833, July, 1987.
- [22] J.M. Cioffi, "The fast QR adaptive filter", submitted to IEEE Trans. ASSP.
- [23] J.M. Cioffi, "The fast Householder filters RLS adaptive filter" , Proc. IEEE ICASSP, pp.1619-1622, Albuquerque, April 1990.
- [24] P. Comon and Y. Robert, "A systolic array for computing BA^{-1} ", IEEE Trans. ASSP, ASSP-35, pp.717-723, June, 1987.
- [25] R.J. Cosentino, "Concurrent error corrections in systolic architecture", IEEE Trans. CAD, Vol 7, pp.117-125, Jan. 1988.

- [26] J.A.B. Fortes and C.S. Raghavendra, "Gracefully degradable processor arrays", IEEE Trans. Computer, Vol C-34, pp.1033-1044, Nov. 1985.
- [27] G.R. Gao and S.J. Thomas, "An optimal parallel Jacobi-like solution method for singular value decomposition", Proc. Int'l Conf. Parallel Processing, pp. 47-53, 1988.
- [28] G.D. de Villiers, "A Gentleman-Kung architecture for finding the singular value of a matrix", Proc. Int'l Conf. Systolic Array, pp.545-554, Ireland, 1989.
- [29] W.M. Gentleman and H.T. Kung, "Matrix triangularization by systolic array," Proc. SPIE, Vol. 298, pp. 298-303, 1981.
- [30] G.H. Golub and C.F. Van Loan, **Matrix Computation**, 2nd edition, Johns Hopkins, 1989.
- [31] S. Haykin, "Radar array processing for angle of arrival estimation", in **Array Signal Processing**, pp.194-292, Haykin, Ed., Prentice-Hall, 1985.
- [32] S. Haykin, **Adaptive Filter Theory**, Prentice-Hall, 1986.
- [33] D.E. Heller and I.C.F. Ipsen, "Systolic networks for orthogonal equivalence transformations and their application", 1982 Conf. Advanced Research in VLSI, M.I.T.
- [34] D.E. Heller and I.C.F. Ipsen, "Systolic networks for orthogonal decomposition", SIAM J. Sci. Stat. Comput. Vol 4, pp.261-269, June, 1983.
- [35] B. Hochet, P. Quinton and Y. Robert, "Systolic Gaussian elimination over GF(p) with partial pivoting", IEEE Trans. Computer, Vol 38, pp.1321-1324, Sep. 1989.
- [36] W. Hoffmann and B.N. Parlett, "A new proof of global convergence for the tridiagonal QL algorithm", SIAM J. Numer. Anal. Vol 15, Oct. 1978.
- [37] S.H. Hosseini, "On fault-tolerant structure, distributed fault-diagnosis, reconfiguration, and recovery of the array processors", IEEE Trans. Computer, Vol 38, pp.932-942, July 1989.
- [38] K.-H. Huang and J.A. Abraham, "Algorithm-based fault-tolerance for matrix operations", IEEE Trans. Computer, Vol C-33, pp.518-528, June, 1984.
- [39] S.F. Hsieh, K.J. R. Liu, and K. Yao, "A fast and effective algorithm for sinusoidal frequency estimation", IEEE Int'l Symposium on Information Theory, San Diego, Jan. 1990.
- [40] S.F. Hsieh, K.J. R. Liu, and K. Yao, "Applications of truncated QR methods to sinusoidal frequency estimation", Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing (ICASSP), pp.2571-2574, Albuquerque, 1990.

- [41] M.G.M. Hussain and M. Jaragh, "A triangular systolic array for the discrete-time deconvolution", IEEE Trans. CAS, Vol 36, pp.622-628, Apr. 1989.
- [42] K. Hwang and F.A. Briggs, **Computer Architecture and Parallel Processing**, McGraw-Hill, 1984.
- [43] I. Ipsen, "Singular value decomposition with systolic array," Proc. SPIE, Vol 495, pp.13-21, 1984.
- [44] S.N. Jean, C.W. Chang and S.Y. Kung, "Graceful degradation schemes for static/dynamic wavefront array", Proc. Int'l Conf. Parallel Processing, pp.249-255, Aug. 1988.
- [45] L. Johnsson, "A computational array for the QR-method," 1982 Conference on Advanced Research in VLSI, M.I.T., pp. 123-129.
- [46] J.-Y. Jou and J.A. Abraham, "Fault-tolerant matrix operation on multiple processor system using weighted checksums", Proc. SPIE Vol 495 Real Time Signal Processing VII, pp.94-101, 1984.
- [47] J.-Y. Jou and J.A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures", Proc. IEEE, Vol 74, pp.732-741, May, 1986.
- [48] J.-Y. Jou and J.A. Abraham, "Fault-tolerant algorithms and architectures for real time signal processing", Proc. Int'l Conf. Parallel Processing, pp.359-362, Aug. 1988.
- [49] S. Kalson and K. Yao, "Systolic array processing for order and time recursive generalized least-squares estimation," Proc. SPIE, Vol. 564, Real Time Signal Processing VIII, pp. 28-38, 1985.
- [50] S.M. Kay, **Modern Spectral Estimation**, Prentice-Hall, 1988.
- [51] I. Koren and M.A. Breuer, "On area and yield considerations for fault-tolerant VLSI processor arrays", IEEE Trans. Computer, Vol C-33, pp.21-27, Jan. 1984.
- [52] K. Konstantinides and K. Yao, "Statistical analysis of effective singular values in matrix rank determination", IEEE Trans. ASSP, ASSP-36, pp.757-736, May 1988.
- [53] I. Koren and D.K. Pradham, "Yield and performance enhancement through redundancy in VLSI and WSI multiprocessor systems", IEEE Proc. Vol 74, pp.699-711, May, 1986.
- [54] H.T. Kung, "Why systolic architectures?", IEEE Computer, Vol 15, pp.37, Jan. 1982.

- [55] H.T. Kung and M.S. Lam, "Wafer-scale integration and two-level pipelined implementation of systolic array", J. Parallel Distrib. Comput., Vol 1, pp.32-63, 1984.
- [56] S.Y. Kung *et al.*, **VLSI and modern signal processing**, Prentice-Hall, 1985.
- [57] S.Y. Kung, C.W. Chang, and C.W. Jen, "Real-time reconfiguration for fault-tolerant VLSI array processors", Proc. Real-Time Systems Symposium, pp.46-54, 1986.
- [58] S.Y. Kung, **VLSI Array Processors**, Prentice Hall, 1988.
- [59] H. Lev-Ari and B. Friedlander, "On the systematic design of fault-tolerant processor arrays with application to digital filter", Proc. IEEE Workshop on VLSI Signal Processing, pp.483-494, Nov. 1988.
- [60] H. Leung and S. Haykin, "Stability of recursive QRD LS algorithms using finite-precision systolic array implementation", IEEE Trans. ASSP, VOL37 pp.760-763, May 1989.
- [61] R.A. Lincoln and K. Yao, "Efficient systolic Kalman filtering design by dependence graph mapping", VLSI Signal Processing III, pp.396-407, Nov. 1988.
- [62] F. Ling, D. Manolakis, and J.G. Proakis, "A recursive modified Gram-Schmidt algorithm for least-squares estimation", IEEE Trans. ASSP, Vol. ASSP-34, pp.829-836, Aug. 1986.
- [63] K.J.R. Liu and K. Yao "On uniform one-chip VLSI design considerations for some discrete orthogonal transforms", Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) pp.2136-2139, New York, April 1988.
- [64] K.J.R. Liu and K. Yao, "A systematic approach to bit recursive systolic array design" Proc. IEEE International Conference on Systolic Array, pp.685-694, San Diego, May 1988.
- [65] K.J.R. Liu and K. Yao, "Gracefully degradable real-time algorithm-based fault-tolerant method for QR recursive least-squares systolic array", in Systolic Array Processors, Ed. McCanny, McWhirter, and Swartzlander, pp. 401-410, Prentice Hall (UK), 1989.
- [66] K.J.R. Liu, S.F. Hsieh, and K. Yao, "Recursive LS filtering using block Householder transformations", Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing, pp.1631-1634, Albuquerque, 1990.
- [67] K.J.R. Liu and K. Yao, "Spectral decomposition via systolic triarray based on QR iteration", Proc. IEEE Int'l Conf. Acoustic, Speech, and Signal Processing (ICASSP), pp.1017-1020, Albuquerque, 1990.

- [68] K.J.R. Liu, "Dynamic range for finite-precision QRD LS algorithm and its stability", Proc. Int'l Sym. Circuits and Systems (ISCAS), pp.3142-3145, New Orleans, May 1990.
- [69] K.J.R. Liu and K. Yao, "Multi-phase systolic architectures for spectral decomposition" Int'l Conf. on Parallel Processing, August, 1990.
- [70] K.J.R. Liu, S.F. Hsieh, and K. Yao, "Two-level pipelined implementation of systolic block Householder transformations with application to RLS algorithm" Int'l Conf. on Application-Specific Array Processors, Princeton, Sep. 1990.
- [71] F.T. Luk, "A parallel method for computing the generalized singular value decomposition", J. Parallel and Distributed Computing 2, pp.250-260, 1985.
- [72] F.T. Luk, "A triangular processor array for computing singular value," Linear Algebra and Its Applications, Vol. 77, pp. 259-273, 1986.
- [73] F.T. Luk, "A rotation method for computing the QR-decomposition", SIAM J. Sci. Stat. Comput. Vol 7, pp.452-459, Apr. 1986.
- [74] F.T. Luk and H. Park, "Fault-tolerant matrix triangulization on systolic array", IEEE Trans. Computer, Vol 37, pp.1434-1438, Nov. 1988.
- [75] F. T. Luk and S. Qiao, "Analysis of a recursive least-squares signal-processing algorithm," SIAM J. Sci. Stat. Comput. Vol. 10, No. 3, pp. 407-418, May 1989.
- [76] V.J. Mathews and Z. Xie, "Fixed-point error analysis of stochastic gradient adaptive lattice filters", IEEE Trans. ASSP, Vol 38, pp.70-80, Jan. 1990.
- [77] J.V. McCanny and J.G. McWhitter, "Some systolic array developments in the United Kingdom", IEEE Computer, Vol 20, pp.51-64, July 1987.
- [78] J.G. McWhirter, "Recursive least-squares minimization using a systolic array", Proc. SPIE, Vol 431, Real Time Signal Processing VI, pp.105-112, 1983.
- [79] J.G. McWhirter and T.J. Shepherd, "An efficient systolic array for MVDR beam-forming," Proc. Int'l Conf. Systolic Array, pp. 11-20, 1988.
- [80] J.G. McWhitter and T.J. Shepherd, "Systolic array processor for MVDR beam-forming", IEE Proc. Vol 135, Pt. F, pp.75-80, 1989.
- [81] D.I. Moldovan, C.I. Wu and J.A.B. Fortes, "Mapping arbitrary large QR algorithm into a fixed size VLSI array", Proc. Int'l Conf. on Parallel Processing, pp.365-373, 1984.
- [82] J.H. Moreno and T. Lang, "A multilevel pipelined processor for the singular value decomposition", Proc. SPIE Vol 698, Real Time Signal Processing IX, 1986.

- [83] J.H. Moreno and T. Lang, "Comments on "A systolic array for computing BA^{-1} ", IEEE Trans. ASSP, Vol 37, pp.1786-1789, Nov. 1989.
- [84] J.G. Nash, K.W. Przytula, and S. Hansen, "Systolic/Cellular processor for linear algebraic operations", VLSI Signal Processing II, pp.306-315, 1986.
- [85] N.L. Owsley, "Sonar array processing", in **Array Signal Processing**, pp.115-193, Haykin, Ed., Prentice-Hall, 1985.
- [86] C.C. Paige, "Computing the generalized singular value decomposition", SIAM J. Sci. Stat. Comput. Vol 7, Oct. 1986.
- [87] B. N. Parlett, "Analysis of algorithms for reflections in bisectors," SIAM Review, Vol. 13, No. 2, pp. 197-208, Apr. 1971.
- [88] S.K. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor arrays", Proc. of IEEE, 76, pp.259-269, March 1988.
- [89] C.P. Rialan and L.L. Scharf, "Cellular architectures for implementing projection operators", IEEE Trans. ASSP, ASSP-35, pp.1619-1627, Nov. 1987.
- [90] W. Robertson and W. Phillips, "A systolic MUSIC system for VLSI implementation", Proc. IEEE ICASSP, pp.2577-2580, 1989.
- [91] R. Roncella and R. Saletti, "A VLSI systolic adder for digital filtering of delta-modulated signal", IEEE Trans. ASSP, Vol 37, pp.749-754, 1989.
- [92] M. Sami and R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays", Proc. IEEE, pp.712-722, May 1986.
- [93] R.O. Schmidt, "A signal subspace approach to multiple emitter location and spectral estimation", Ph.D. dissertation, Stanford Univ. 1981.
- [94] R. Schreiber, "Systolic array for eigenvalue computation", Proc. SPIE Vol 341, Real Time Signal Processing V, 1982.
- [95] R. Schreiber, "Systolic linear algebra machines in digital signal processing", in **VLSI and Modern Signal Processing**, pp.389-405, S.Y. Kung et al., Eds., Prentice-Hall, 1985.
- [96] R. Schreiber, "Solving eigenvalue and singular value problems on an undersized systolic systolic array," SIAM J. Sci. Stat. Comput. Vol 7, pp.441, Apr. 1986.
- [97] R. Schreiber, "Implementation of adaptive array algorithms", IEEE Trans. ASSP Vol ASSP-34, pp.1038-1045, Oct. 1986.

- [98] K.G. Shin and T.-H. Lin, "Modeling and measurement of error propogation in a multimodule computing system", IEEE Trans. Computer, Vol 37, pp.1053-1066, Sep. 1988.
- [99] A. Steinhardt, "Householder transformation," IEEE ASSP Mag., 1988.
- [100] G.W. Stewart, "Incorporating origin shifts into the QR algorithm for symmetric tridiagonal matrices", Comms. ACM, Vol 13, June, 1970.
- [101] G.W. Stewart, **Introduction to Matrix Computations**, Academic Press, 1973.
- [102] R.A. Thisted, **Elements of statistical computing**, Chapman and Hall, 1988.
- [103] N. Torralba and J.J. Navarro, "Size-independent systolic algorithms for QR iteration and Hessenberg reduction", Proc. Int'l Conf. Systolic Array, pp.166-175, 1989.
- [104] N.-K. Tsao, "A note on implementing the Householder transformation," SIAM J. Numer. Anal., Vol. 12, No. 1, pp. 53-58, Mar. 1975.
- [105] B. Van Veen, "Systolic processors for linearly constrained beamforming", IEEE Trans. ASSP, Vol 37, pp.600-604, Apr. 1989.
- [106] C.R. Ward, P.J. Hargrave and J.G. McWhirter, "A novel algorithm and architecture for adaptive digital beamforming", IEEE Trans. Antennas Propagat., Vol AP-34, pp.338-346, March, 1986.
- [107] J.H. Wilkinson, **Algebraic Eigenvalue Problem**, Oxford University Press, 1965.
- [108] J.H. Wilkinson, "Global convergence of tridiagonal QR algorithm with origin shift", Linear Algebra and Its Applications 1, pp.409-420, 1968.
- [109] J. H. Wilkinson, "Modern error analysis," SIAM Review, Vol. 13, No. 4, pp. 548-568, Oct. 1971.